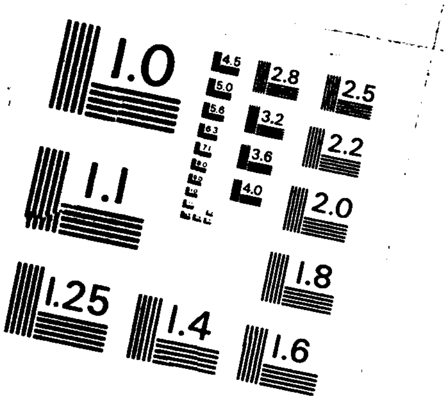
AD-A166 848   A COHERENT VLSI DESIGN ENVIRONMENT(U) MASSACHUSETTS   1/1
INST OF TECH CAMBRIDGE P PENFIELD 31 MAR 86
N00014-80-C-0622
UNCLASSIFIED   F/G 9/5   NL


1.0
4.5
5.0
5.6
6.3
7.1
8.0
9.0
10
2.8
3.2
3.6
4.0
2.5
2.2
2.0
1.1
1.8
1.25
1.4
1.6

12

MASSACHUSETTS INSTITUTE OF TECHNOLOGY VLSI PUBLICATIONS

A COHERENT VLSI DESIGN ENVIRONMENT

Semiannual Technical Report for the period October 1, 1985 to March 31, 1986

Massachusetts Institute of Technology
Cambridge, MA 02139

Principal Investigators:	Paul Penfield, Jr.	(617) 253-2506
	Lance A. Glasser	(617) 253-4677
	Thomas F. Knight, Jr.	(617) 253-7807
	Charles E. Leiserson	(617) 253-5833
	Ronald L. Rivest	(617) 253-5880
	John L. Wyatt, Jr.	(617) 253-6028

This research was sponsored by Defense Advanced Research Projects Agency (DoD), through the Office of Naval Research under Contract N00014-80-C-0622.

Microsystems
Research Center
Room 39-321

Massachusetts
Institute
of Technology

Cambridge
Massachusetts
02139

Telephone
(617) 253-8138

TABLE OF CONTENTS

Selected Publications (starting after page 9)

P. D. Bassett, "A High-Speed Asynchronous Communication Technique for MOS Systems," M.S. and E.E. thesis, Department of Electrical Engineering and Computer Science, May 1985. Also MIT VLSI Memo No. 85-283, December 1985.*

A. T. Ishii, "Interprocessor Communication Issues in Fat-Tree Architectures," B.S. thesis, Department of Electrical Engineering and Computer Science, MIT, May 1985. Also MIT VLSI Memo No. 85-268, October 1985.*

B. M. Maggs, "Computing Minimum Spanning Trees on a Fat-Tree Architecture," B.S. thesis, Department of Electrical Engineering and Computer Science, MIT, May 1985. Also MIT VLSI Memo No. 85-269, October 1985.*

C. A. Phillips, "Space-Efficient Algorithms for Computational Geometry," M.S. thesis, Department of Electrical Engineering and Computer Science, MIT, August 1985. Also MIT VLSI Memo No. 85-270, October 1985.*

R. I. Greenberg and C. E. Leiserson, "Randomized Routing on Fat-Trees," Proceedings, Twenty-Sixth Annual Symposium on Foundations of Computer Science, Portland, OR, IEEE Computer Society, pp. 241-249, October 21-23, 1985. Also MIT VLSI Memo No. 85-260, September 1985.*

T.-A. Chu and L. A. Glasser, "Synthesis of a Self-timed Controller for a Successive-approximation A/D Converter," MIT VLSI Memo No. 85-274, October 1985.

L. A. Glasser, "Synchronizer Failure in A/D Converters," MIT VLSI Memo No. 85-276, October 1985.

C. E. Leiserson, "Fat-trees: Universal Networks for Hardware-efficient Supercomputing," IEEE Transactions on Computers, vol. C-34, no. 10, October 1985, pp. 892-901. An early version appeared in the 1985 International Conference on Parallel Processing.

P. O'Brien and J. L. Wyatt, Jr., "Signal Delay in Leaky RC Mesh Models for Bipolar Interconnect," MIT VLSI Memo No. 85-278, November 1985.*

P. R. O'Brien and J. L. Wyatt, Jr., "Signal Delay in ECL Interconnect," to appear in Proceedings, 1986 International Symposium on Circuits and Systems, Santa Clara, CA, May 1986. Also MIT VLSI Memo No. 86-296, February 1986.

----

* Abstract only. Complete version available from Microsystems Research Center, Room 39-321, MIT, Cambridge, MA 02139; telephone (617) 253-8138.

RESEARCH OVERVIEW

This report covers the period from October 1, 1985 through March 31, 1986. The research discussed here is described in more detail in several published and unpublished reports cited below.

The CAD frame Schema has received limited but instructive applications. Connections to remote simulation servers are transparent to the user. More substantial uses are anticipated during the next half year.

Waveform bounding results have been extended to ECL technology, and a variety of results are being applied to an analog application in early vision. Many of the previous results have been transferred to industry and are being used in commercial simulators.

Fundamental speed limits of collections of active devices with parasitics have been found. These are independent of how the devices are interconnected. A reliability simulator has been written, and models are being developed for it, to describe three important effects, namely metal migration, hot-electron trapping, and time-dependent dielectric breakdown.

Work continues on the fat-tree architecture, and on the efficient realization of various classes of circuits. New results in graph bisection and routing were made during this period. It is now known how to imbed some classes of communications networks in others, for example binary trees in hypercubes.

THE DESIGN OF SCHEMA

We have been making steady progress on Schema over the past few months. The bulk of our energies have been devoted to fixing bugs and providing missing functionality for designers. We expect this effort to continue for another four or five months. It is worthwhile recalling what "basic" functionality Schema is providing. It contains a flexible schematic-entry system, capable of dealing with multilevel hierarchical schematics. It has been used for MOS, TTL and analog designs and provides the canonical interface to the topological structures used by the analysis routines.

A simulation package allows the designer to use a transient simulator like Spice to simulate a circuit. The simulation may be run on the local machine or remotely on a VAX. In either case the designer's interface is identical and is completely graphics oriented. This system is currently being extended to include other types of simulators.

A simple layout system, along the lines of DPL/Daedalus, can be used to create and edit masks. As in the schematic system, the ultimate description generated by Schema is a procedure that when run will create the mask. This provides the ultimate in parameterizability. Structures have been set up to allow the system to be used for NMOS, CMOS and GaAs designs.

All of these tools and their data structures reside in a common address space, so it is easy for a CAD designer to build programs that straddle different design domains. Furthermore, the data structures used by Schema have common interface specifications, so they can be mixed quite flexibly.

The two designs we have run through Schema were a CMOS sorter and a TTL PC board for gathering statistics from a Multibus. Neither design was completed using Schema, but at least in the CMOS case, the designer was able to complete about 95% of the design using Schema. Both efforts gave us a great deal of information about what is missing in the system and what performance improvements are needed. These efforts are under way now and we expect to try a more substantial project towards the end of the spring.

THE WAVEFORM BOUNDING APPROACH TO DELAY ESTIMATION

In the past six months we have tightened the bounds for interconnect delay, we have begun to resolve the modeling issues that arise when the delay bounds are used for timing analysis of ECL chips, and we have initiated an analytical study and preliminary design effort to determine the feasibility of doing analog or hybrid analog/digital massively parallel signal processing in MOS VLSI for early vision applications.

Dave Standley is doing the work on bound-tightening for his S.M. thesis. He has written a very elegant derivation of the new time constant T_{Pi} for arbitrary trees, and his result is now in use in the new line of CAD products just released by Tangent Systems (2840 San Tomas Expressway, Suite 200, Santa Clara, CA 95051). He has also found a new lower bound for the rising step response of RC trees. This research area is becoming mature, and all the easy results are probably known now.

Peter O'Brien is doing the work on interconnect delay in ECL for his S.M. thesis. The first part consisted of extending the Penfield results to include the case of "leaky" RC trees, i.e., trees with resistive paths to ground, required for modeling bipolar transistors as loads. This work is complete. He is now working on the second part of the problem, developing the required macromodels for ECL logic gates. Both parts of this project are being done in close collaboration with the Digital Equipment Corporation for use in an inhouse ECL timing analyzer under development there.

The early vision work began with an attempt to apply our interconnect delay results to a proposed analog early vision chip to estimate how fast it might run. Discussions with Prof. Thomasio Poggio and Dr. Christof Koch in the MIT AI Lab have led to a joint interest in using the properties of RC networks, realized by a rectangular grid of polysilicon on an MOS chip, to perform a variety of early vision tasks quite rapidly and cheaply by analog methods. The simplest and most promising project seems to be analog depth interpolation for stereo vision. The idea is quite speculative but the potential improvement over current implementations such as the connection machine is many orders of magnitude in size and cost, and one or two orders of magnitude in speed.

HIGH PERFORMANCE CIRCUIT DESIGN

RELIC, our reliability simulator, is beginning to show signs of life. The objective of this research effort is to demonstrate a simulator which predicts the reliability of MOS VLSI circuits. More precisely, we would like to predict and compare the reliability of different circuits with respect to those failure mechanisms under the control of the circuit designer. In this project we are examining three failure mechanisms: metal migration, hot- electron trapping, and time-dependent dielectric breakdown (TDDB). The long term failure statistics of the circuit are predicted based on the "stress" accumulated during one cycle, assuming that this cycle repeats forever.

There are three parts to the simulator: the preprocessor, the models, and the post processor. Work has begun on the first two and we hope to have all parts running by June. The models are the central part of the program. We have added 1986 TDDB and metal migration models to the underlying simulator, RELAX, from Berkeley. (Our thanks to Berkeley for letting us have early access to their software.) The hot-electron model is still fighting us.

We are starting a new research project into the high frequency behavior of VLSI circuits. About 20 years ago it was shown that fundamental frequency domain quantities such as f_{max} could be used to predict, for instance, the properties of ring oscillators. We are extending this classical theory in three directions. First, we have reformulated it in terms of Tellegen's theorem. Second, we are extending it to much more complicated -- and realistic -- models. And third, we are making some headway on a nonlinear formulation of the problem. It is too early to predict where all of this will go, but we hope to be able to say things about what sort of performance one could possibly get out of (digital?) networks made up of certain types of components. A possible application for this work is cross comparisons of technologies, where one would like to compare the cleverness of the device engineers rather than that of the circuit designers. Another possibility would be in the area of making technology trade-offs in the design of CMOS devices. For instance, is it "better" to have a larger lateral diffusion with more overlap capacitance but shorter transistors or is it better to minimize the parasitic capacitance?

ARCHITECTURAL DESIGN

Professor Charles E. Leiserson has extended his work on fat-trees and now has improved designs for area- and volume-universal networks. One simple design contains only n small switches for a network with n processors. He and Ron Greenberg have been able to provide good message-routing algorithms for this and other universal networks.

Bruce M. Maggs and Professor Leiserson have been exploring parallel algorithms for fat-trees. The discovery of volume-universal networks with different topologies from fat-trees motivated them to abstract a model for such networks that would allow them to design parallel algorithms, but which would not take advantage of the specific topology of the network. Their new model, called a distributed random-access machine (DRAM), allows the communication requirements of an algorithm to be measured.

Joe Kilian, Shlomo Kipnis, and Professor Leiserson have been investigating the power of multipin nets. Surprisingly, it is possible to make a barrel shifter of n chips, each with $O(\sqrt{n})$ pins, which can perform any shift in 1 clock tick. With 2-pin nets, $n - 1$ pins per chip are required. They can also show that $O(\sqrt{n})$ pins suffice to realize any abelian group of permutations of n elements, and that $O(\sqrt{(n \log n)})$ pins suffice to realize almost all permutations.

Andrew Goldberg has been working on parallel algorithms for network flow problems. He has discovered a new algorithm for maximum flow that should run well on a parallel machine, and in particular, on a DRAM. He and Bob Tarjan of Princeton have been able to convert this parallel algorithm into a normal sequential algorithm that is faster than previously known algorithms for this well-studied problem.

Thang Bui completed his Ph.D. thesis on graph-bisection algorithms. Included in the thesis is joint work with Profs. Leighton and Sipser on an algorithm which is guaranteed to find the optimal bisection for almost all graphs with small bisection width. The algorithm is particularly well suited to low degree graphs such as those that arise in VLSI placement and routing problems. Prof. Leighton and Bui also discovered a method of speeding up and improving the performance of other graph bisection heuristics such as Kernighan-Lin and simulated annealing. Tests and analysis of the new method are still under way, but it now appears that the speed of the heuristics can be cut in half and that the size of the bisections produced can be decreased by 10-25%. These figures represent substantial savings for large graphs.

Bonnie Berger completed her Master's thesis on channel routing. Included in the thesis is joint work with Prof. Leighton on the development of an algorithm which can route any 2-point net channel routing problem using $d + O(\sqrt{d})$ tracks, where d is the density of the channel. The algorithm uses unit vertical overlaps of wires and knock-knees in two layers, but does not otherwise allow wires to overlap. The algorithm is very close to optimal since d tracks are always required for a problem with density d. (Even when arbitrary overlap is allowed, Leighton has shown that most interesting problems still require d layers.) Moreover, the algorithm uses only half as many tracks as the best previously known algorithm, which was discovered by Rivest, Baratz and Miller. For multipoint net problems, the algorithm uses $2d + O(\sqrt{d})$ tracks, which is also half as many as the best previously known algorithms. Professor Leighton is currently working with Berger and others to extend the algorithm to handle multilayer channel-routing problems. The initial results appear quite favorable. They have shown that any L-layer problem can be routed in $d/L + O(\sqrt{d})$ tracks which is very close to the lower bound of d/L tracks. For multipoint-net problems, the bound is $d/(L-1) + O(\sqrt{d})$.

In the area of wafer-scale integration, Prof. Leighton and Peter Shor have shown how to use matching algorithms to construct two-dimensional systolic arrays from the working cells on an N-cell wafer using wires of length $O(\log^{3/4}N)$. The result assumes that the faulty cells are located at random on the wafer, and solves a long-open question in mathematical statistics. The work will be reported in the upcoming ACM Symposium on Theory of Computing in San Francisco.

Prof. Leighton has also made advances on several network-embedding problems. Working with others, he has shown how to efficiently embed arbitrary binary trees in the hypercube and other networks. Although the results are still preliminary, this work will probably include the discovery of the first bounded-degree network known to contain all binary spanning trees as subgraphs. Professor Leighton is also studying related graph-embedding problems with the aim of using one network (e.g., a hypercube) to efficiently simulate another network (e.g., a mesh).

PUBLICATIONS LIST

P. D. Bassett, "A High-Speed Asynchronous Communication Technique for MOS Systems," M.S. and E.E. thesis, Department of Electrical Engineering and Computer Science, May 1985. Also MIT VLSI Memo No. 85-283, December 1985.

A. T. Ishii, "Interprocessor Communication Issues in Fat-Tree Architectures," B.S. thesis, Department of Electrical Engineering and Computer Science, MIT, May 1985. Also MIT VLSI Memo No. 85-268, October 1985.

F. T. Leighton and C. E. Leiserson, "Wafer-Scale Integration of Systolic Arrays," IEEE Transactions on Computers, vol. C-34, no. 5, pp. 448-461, May 1985. Also MIT VLSI Memo No. 85-272, October 1985.

B. M. Maggs, "Computing Minimum Spanning Trees on a Fat-Tree Architecture," B.S. thesis, Department of Electrical Engineering and Computer Science, MIT, May 1985. Also MIT VLSI Memo No. 85-269, October 1985.

C. A. Phillips, "Space-Efficient Algorithms for Computational Geometry," M.S. thesis, Department of Electrical Engineering and Computer Science, MIT, August 1985. Also MIT VLSI Memo No. 85-270, October 1985.

R. I. Greenberg and C. E. Leiserson, "Randomized Routing on Fat-Trees," Proceedings, Twenty-Sixth Annual Symposium on Foundations of Computer Science, Portland, OR, IEEE Computer Society, pp. 241-249, October 21-23, 1985. Also MIT VLSI Memo No. 85-260, September 1985.

T.-A. Chu and L. A. Glasser, "Synthesis of a Self-timed Controller for a Successive-approximation A/D Converter," MIT VLSI Memo No. 85-274, October 1985.

L. A. Glasser, "Synchronizer Failure in A/D Converters," MIT VLSI Memo No. 85-276, October 1985.

C. E. Leiserson, "Fat-trees: Universal Networks for Hardware-efficient Supercomputing," IEEE Transactions on Computers, vol. C-34, no. 10, October 1985, pp. 892-901. An early version appeared in the 1985 International Conference on Parallel Processing.

G. C. Clark and R. E. Zippel, "Schema: An Architecture for Knowledge Based CAD," Digest of Technical Papers, IEEE International Conference on Computer-Aided Design, ICCAD-85, Santa Clara, CA, pp. 50-52, November 18-21, 1985. Also MIT VLSI Memo No. 85-271, October 1985.

P. O'Brien and J. L. Wyatt, Jr., "Signal Delay in Leaky RC Mesh Models for Bipolar Interconnect," MIT VLSI Memo No. 85-278, November 1985.

J. L. Wyatt, Jr., C. A. Zukowski, and P. Penfield, Jr., "Step Response Bounds for Systems Described by M-Matrices, with Application to Timing Analysis of Digital MOS Circuits," Proceedings, 24th IEEE Conference on Decision and Con-

trol, Ft. Lauderdale, FL, pp. 1552-1557, December 11-13, 1985. Also MIT VLSI Memo No. 85-257, September 1985.

B. Berger, M. Brady, D. Brown and T. Leighton, "An Almost Optimal Algorithm for Multilayer Channel Routing," unpublished manuscript, December 1985.

S. Bhatt, F. Chung, T. Leighton and A. Rosenberg, "Optimal Embeddings of Binary Trees in the Boolean Hypercube," unpublished manuscript, December 1985.

B. Berger, "New Upper Bounds for Two-Layer Channel Routing," M.S. thesis, Department of Electrical Engineering and Computer Science, MIT, January 1986.

T. Bui, "Graph Bisection Algorithms," Ph.D. thesis, Department of Electrical Engineering and Computer Science, MIT, January 1986.

B. Chor, C. E. Leiserson, R. L. Rivest, and J. Shearer, "An Application of Number Theory to the Organization of Raster-Graphics Memory," *JACM*, vol. 33, no. 1, pp. 86-104, January 1986. An early version by the first three authors appeared in *Proceedings, Twenty-Third Annual Symposium on Foundations of Computer Science*, Chicago, IL, IEEE Computer Society, pp. 92-99, November 1982.

A. Goldberg and R. E. Tarjan, "A New Approach to the Maximum Flow Problem," to appear in the *18th ACM Symposium on Theory of Computing*, May 1986.

T. Leighton and P. Shor, "Tight Bounds for Minimax Grid Matching, with Applications to the Average Case Analysis of Algorithms," to appear in the *18th ACM Symposium on Theory of Computing*, May 1986.

P. R. O'Brien and J. L. Wyatt, Jr., "Signal Delay in ECL Interconnect," to appear in *Proceedings, 1986 International Symposium on Circuits and Systems*, Santa Clara, CA, May 1986. Also MIT VLSI Memo No. 86-296, February 1986.

T. Leighton and C. E. Leiserson, "A Survey of Algorithms for Integrating Wafer-Scale Systolic Arrays," *VLSI Circuit and Architecture Design*, E. Swartzlander, Jr., ed., Marcel-Dekker, Inc., 1986.

T. Bui, S. Chaudhuri, T. Leighton and M. Sipser, "Graph Bisection Algorithms with Good Average Case Behavior," to appear in *Combinatorica*, .

F. Chung, T. Leighton and A. Rosenberg, "Embedding Graphs in Books: A Layout Problem With Applications to VLSI Design," to appear in *SIAM J. Algebraic and Discrete Methods*, .

T. Leighton and A. Rosenberg, "Three-Dimensional Circuit Layouts," to appear in *SIAM J. Computing*, .

C. E. Leiserson and B. M. Maggs, "Communication-efficient Parallel Graph Algorithms," submitted for publication.

TALKS WITHOUT PROCEEDINGS

F. T. Leighton, "Wafer-Scale Integration of Systolic Arrays," Columbia University Theory Day, September 1985, and IFIP Workshop on Wafer-Scale Integration, Grenoble France, March 1986.

F. T. Leighton, "The Expected Behavior of Assignment Algorithms," Cornell Computer Science Department, November 1985, and AT&T Bell Laboratories (Holmdel) Computer Science Division, December 1985.

F. T. Leighton, "Networks, Parallel Computation and VLSI Design," MAA Annual Regional Meeting, Philadelphia, November 1985, and Fordham University Computer Science Department, February 1986.

B. A. Berger and T. Leighton, "New Bounds and Algorithms for Channel Routing," MIT Fall 1985 VLSI Research Review, Cambridge, MA, December 16, 1985.

M. St. Pierre, "A Simulation Environment for Schema," MIT Fall 1985 VLSI Research Review, Cambridge, MA, December 16, 1985.

J. L. Wyatt, Jr., "Signal Propagation Delay in Linear RC Models for On-Chip Interconnect," Electromagnetic Wave Theory Group, Dept. of Electrical Engineering and Computer Science, MIT, February 1986.

P. Penfield, Jr., "Computer-Aided Fabrication and Other VLSI Research at MIT," IBM Corporation, Yorktown Heights, NY, March 21, 1986.

DEPARTMENT OF ELECTRICAL ENGINEERING AND COMPUTER SCIENCE

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

CAMBRIDGE, MASSACHUSETTS 02139

VLSI Memo No. 85-283 December 1985

A High-Speed Asynchronous Communication Technique for MOS VLSI Systems*

Paul D. Bassett**

ABSTRACT

As MOS technologies advance, the relative differences between on-chip and off-chip delays increase. Drivers and receivers can be designed which allow high bit rate communications (>100 Mbits/sec) between MOS chips at the expense of increased latency. Designing synchronous systems which couple a high clock frequency with large and variable delays is difficult and expensive due to the complexity of insuring that no delays violate the constraints imposed by synchronous operation.

A circuit-based technique for automatically adjusting signal delays in an MOS system has been developed. The Dynamic Delay Adjustment (DDA) technique provides reliable high speed communications directly between MOS chips independent of the delay between the chips. The amount of phase jitter immunity provided by the synchronizer can be traded off against circuit complexity; the signal delays are adjusted continuously to track temperature induced delay variations. A 3 micron CMOS DDA synchronizer has been fabricated to confirm the validity of the DDA approach; test results will be presented.

*Submitted to the Department of Electrical Engineering and Computer Science on May 14, 1985 in partial fulfillment of the requirements for the degree of Master of Science. This work was sponsored in part by an RCA Fellowship and in part by the Defense Advanced Research Projects Agency under Contract number N00014-80-C-0622.

**Current address: Bolt, Beranek and Newman, Room 6/501, 10 Molton St., Cambridge, MA 02238; (617) 497-2471.

DEPARTMENT OF ELECTRICAL ENGINEERING AND COMPUTER SCIENCE

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

CAMBRIDGE, MASSACHUSETTS 02139

VLSI Memo No. 85-268 October 1985

Interprocessor Communication Issues in Fat-Tree Architectures*

Alexander Toichi Ishii**

ABSTRACT

In recent years, it has become increasingly evident that conventional computer architectures will be unable to perform, in an acceptable time frame, many of the computational functions that we would desire of them. Consequently, much research has been devoted to the concept of constructing super-computers, which will be able to exploit the potential for parallel computation intrinsic to many large computational problems.

Recently, Leiserson has proposed a multiprocessor scheme based on Leighton's "tree of meshes," called a "fat-tree." Conceptually, such a multiprocessor would be comprised of a set of n processing elements each situated as a leaf in a complete binary tree. Internal nodes would be high speed switches which route messages being passed between processing elements, while edges between nodes would be bundles of constant bandwidth communication paths.

The purpose of this document will be to address and define some of the issues which effect interprocessor communication within a fat-tree multiprocessor. Specifically, we will cover the following topics:

1. Addressing in a fat-tree
2. Generation of addresses in a fat-tree
3. Allocation of communication resources in a fat-tree

*Submitted to the Department of Electrical Engineering and Computer Science in partial fulfillment of the requirements for the degree of Bachelor of Science on May 10, 1985. This research was supported in part by the Defense Advanced Research Projects Agency under contract number N00014-80-C-0622.

**Department of Electrical Engineering and Computer Science, MIT, Room NE43-313, Cambridge, MA 02139, (617) 253-7843.

DEPARTMENT OF ELECTRICAL ENGINEERING AND COMPUTER SCIENCE

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

CAMBRIDGE, MASSACHUSETTS 02139

VLSI Memo No. 85-269 October 1985

Computing Minimum Spanning Trees On a Fat-Tree Architecture*

Bruce M. Maggs**

ABSTRACT

This paper presents two algorithms for computing the minimum spanning forest of an input graph on a fat-tree architecture. One algorithm is deterministic, and the other probabilistic. The deterministic algorithm generates $O(\log^3|V|)$ message sets, each of which can be delivered in $O(\beta(G))$ delivery cycles. The probabilistic algorithm generates $O(\log^2|V|)$ message sets, each of which can be delivered in $O(\beta(G))$ delivery cycles.

*Submitted to the Department of Electrical Engineering and Computer Science, MIT, in partial fulfillment of the requirements for the degree of Bachelor of Science, May, 1985. This research was supported in part by the Defense Advanced Research Projects Agency under contract number N00014-80-C-0622.

DEPARTMENT OF ELECTRICAL ENGINEERING AND COMPUTER SCIENCE

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

CAMBRIDGE, MASSACHUSETTS 02139

VLSI Memo No. 85-270 October 1985

Space-Efficient Algorithms for Computational Geometry*

Cynthia A. Phillips**

ABSTRACT

This thesis presents an algorithm for determining the connectivity of a set of N rectangles in the plane, a problem central to avoiding aliasing in VLSI design rule checkers. Previous algorithms for this problem either worked slowly with a small amount of primary memory space, or worked quickly but used more space. The algorithm presented here, based upon a technique called scanning, operates in O(N lg N) time in the worst case. This matches the running time of the best known sequential algorithm for this problem. Because we use a machine model that explicitly incorporates secondary memory, the new connected components algorithm avoids unexpected disk thrashing which leads to lower performance. The algorithm uses O(W) primary memory space, where W, the scan width, is the maximum number of rectangles to cross any vertical cut. It requires no more than O(N) transfers between primary and secondary memory.

When a vertical line passes through a set of rectangles, those rectangles cut by the line form a set of line segments. The key to development of space-efficient algorithms using a two layer memory model is that appropriate manipulations of these segments alone can solve more complicated problems such as the connected components problem. This thesis introduces interval trees, a simple, sparse, data structure for storing a set of k line segments. With this data structure, a variation on a balanced search tree, one can perform each of the following operations in O(lg k) time in the worst case: 1) insert a new segment, 2) delete a segment, and 3) given a test interval, return a segment which intersects that test interval or return nil if there is no such segment. This data structure is used in the new connected components algorithm. It can also be used to improve other existing algorithms for computational geometry problems.

*Submitted to the Department of Electrical Engineering and Computer Science on August 30, 1985 in partial fulfillment of the requirements for the degree of Master of Science. This work was supported by the Defense Advanced Research Projects Agency under contract number N00014-80-C-0622.

**Department of Electrical Engineering and Computer Science, MIT, Room NE43-834, Cambridge, MA 02139, (617) 253-2345.

DEPARTMENT OF ELECTRICAL ENGINEERING AND COMPUTER SCIENCE

MASSACHUSETTS INSTITUTE OF TECHNOLOGY
CAMBRIDGE, MASSACHUSETTS 02139

VLSI Memo No. 85-260 September 1985

Randomized Routing on Fat-Trees*

Ronald I. Greenberg and Charles E. Leiserson**

ABSTRACT

Fat-trees are a class of routing networks for hardware-efficient parallel computation. This paper presents a randomized algorithm for routing messages on a fat-tree. The quality of the algorithm is measured in terms of the load factor of a set of messages to be routed, which is a lower bound on the time required to deliver the messages. We show that if a set of messages has load factor $\lambda = \Omega(\lg n \lg \lg n)$ on a fat-tree with n processors, the number of delivery cycles (routing attempts) that the algorithm requires is $O(\lambda)$ with probability $1-O(1/n)$. The best previous bound was $O(\lambda \lg n)$ for the off-line problem where switch settings can be determined in advance. In a VLSI-like model where hardware cost is equated with physical volume, we use the routing algorithm to demonstrate that fat-trees are universal routing networks in the sense that any routing network can be efficiently simulated by a fat-tree of comparable hardware cost.

*This research was supported in part by the Defense Advanced Research Projects Agency under Contract number N00014-80-C-0622. Greenberg is supported in part by a Fannie and John Hertz Foundation Fellowship. Leiserson is supported in part by an NSF Presidential Young Investigator Award.

**Laboratory for Computer Science, MIT, Cambridge, MA 02139, Greenberg: Room NE43-309, (617) 253-5883; Leiserson: Room NE43-835, (617) 253-5833.

DEPARTMENT OF ELECTRICAL ENGINEERING AND COMPUTER SCIENCE

MASSACHUSETTS INSTITUTE OF TECHNOLOGY
CAMBRIDGE, MASSACHUSETTS 02139

VLSI Memo No. 85-274

October 1985

Synthesis of a Self-timed Controller for a Successive-approximation A/D Converter*

Tam-Anh Chu and Lance A. Glasser**

ABSTRACT

This paper documents the procedure for designing a self-timed controller for a successive-approximation A/D converter. From the functional specification, a Signal Transition Graph is constructed to describe the operation of the control circuit. This graph is then modified into a well-formed graph. Such a graph can be transformed into a deadlock-free and hazard-free implementation directly. The structure of the control circuit and the logic equations are then derived directly from the graph.

*This research was supported in part by a Hughes Fellowship, by the National Science Foundation under grant number MCS-7915255, and by the Defense Advanced Research Projects Agency under contract number N00014-80-C-0622.

**Department of Electrical Engineering and Computer Science, MIT, Cambridge, MA 02139. Chu: Room NE43-237, (617) 253-8861; Glasser: Research Laboratory of Electronics, Room 36-880, (617) 253-4677.

Synthesis of a Self-timed Controller for a Successive-approximation A/D Converter

Tam-Anh Chu[1]
Department of EECS

Lance A. Glasser[2]
Department of EECS and the Research Laboratory of Electronics
Massachusetts Institute of Technology
Cambridge, Massachusetts 02139

Abstract

This paper documents the procedure for designing a self-timed controller for a successive-approximation A/D converter. From the functional specification, a Signal Transition Graph is constructed to describe the operation of the control circuit. This graph is then modified into a well-formed graph. Such a graph can be transformed into a deadlock-free and hazard-free implementation directly. The structure of the control circuit and the logic equations are then derived directly from the graph.

1. Introduction

This paper describes the design of a control circuit for a successive-approximation A/D converter. This controller is an asynchronous self-timed circuit in which all control actions are carried out through the use of the Request/Acknowledge signaling protocol. From a functional specification of the control circuit, a Signal Transition Graph (STG) is constructed to describe the behavior of the circuit. It is then modified into a well-formed graph, which is one satisfying the liveness and persistency properties. This

[1] This research was supported in part by a Hughes Fellowship and the NSF under grant number MCS-7915255.

[2] This research was supported by the Defense Advanced Research Project Agency under contract number N00014-80-C-0622.

STG can be converted directly into a logic circuit through a number of synthesis steps. A theory of the Signal Transition Graph model is discussed in [1]. Sufficient details about the model will be given here to explain the synthesis process from STG.

The main advantage of this realization method is that it can produce a circuit directly from a well-formed specification, and the circuit is speed-independent, i.e., it operates correctly with any combination of delays of logic gates. This feature is important for VLSI applications, as it is inefficient and not always possible to fine-tune the delays of logic gates on chip to make an asynchronous system work. The STG model allows the specification of *concurrency*, and hence the control logic synthesized from this model supports concurrent operations. The traditional approach for designing asynchronous state machines can only model sequential control actions, and furthermore they are difficult to realize because of problems due to races and hazards. In contrast, the approach presented here produces hazard-free control circuits capable of handling parallel operations.

2. Behavior Specification of the Controller

The block diagram of the successive approximation A/D converter is shown in Figure 1. The input comparator compares the input voltage v_{in} and the reference voltage v_{ref} and produces a digital 1-bit result. The comparator has a control input Z_r which zeroes it when Z_r makes a low-to-high transition, and starts the comparison when Z_r makes a high-to-low transition. It also has a *mutual-exclusion* circuit whose output is active (=1) only when the comparator output is valid. This is required because the comparison time is a function of the difference between the input voltage and the reference voltage; the smaller the difference, the longer the time it takes for the comparator to decide. This is the familiar problem of metastability[2].

The latch and the combinational logic form a finite state machine performing the successive approximation algorithm. Note that this machine operates in *pulse mode* and is not the same as the self-timed controller we are synthesizing. Due to the fact that this machine performs many data-dependent operations, it is more economical and straight-forward to implement it in this form instead of a Huffman asynchronous state machine. Data are latched on the rising transition of signal L_r and held in registers

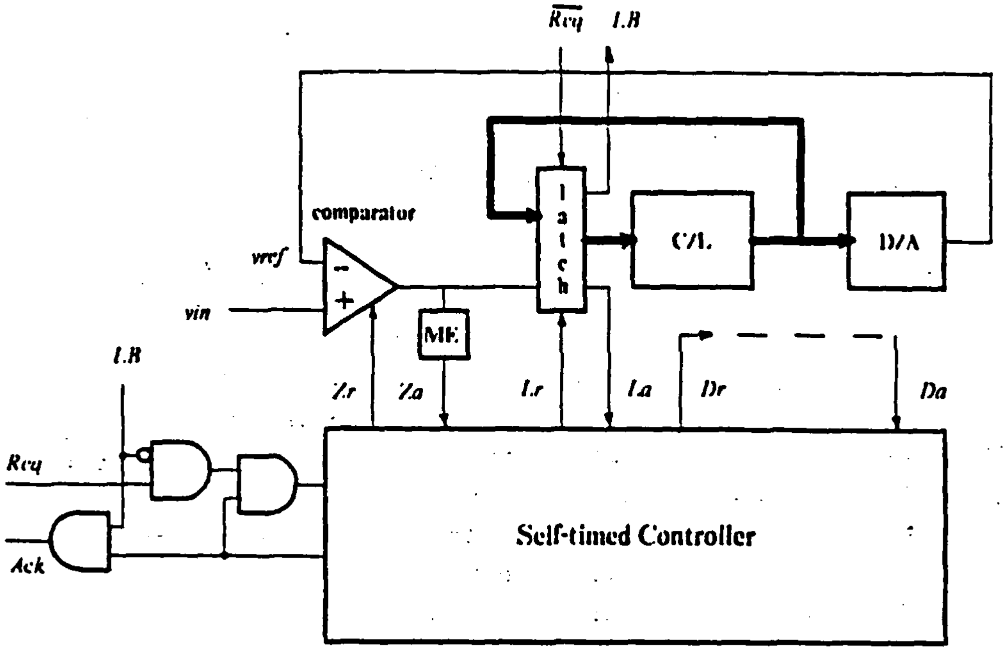


Figure 1: Block diagram of the successive-approximation A/D converter

after L_r goes low. Signal L_a goes high as soon as data are latched, and goes low shortly after L_r goes low. The reset input of the latch is controlled by signal $\overline{Req}$, so that when Req is low, its outputs are reset to the appropriate initial state. Signal LB is the *Last-Bit* signal which goes high when the converter has determined the last bit of the digital word. The D/A converter at the right of the diagram accepts the digital word produced by the state machine and generates the analog voltage v_{ref}. The combined delay of the combinational logic and the D/A converter is matched by some delay circuit as indicated by a dashed line from D_r to D_a. While it is possible to accomplish this timing constraint in a speed-independent manner using dual rail coding, a simple delay circuit is more justifiable from an engineering standpoint.

Initially, the state of the system is $Req = Ack = Z_r = Z_a = L_r = L_a = 0$ and $D_r = D_a = 1$. Since $Req = 0$, the latch is initialized with $LB = 0$. Thus, the and-gate whose input is Req is enabled and the and-gate whose output is Ack is disabled. When Req is raised, Z_r will go high and initiate a cycle of the successive-approximation algorithm. After each cycle, D_a will restart another cycle until LB becomes high during the last cycle of the algorithm. In the last cycle, *Ack* is raised when D_a goes high. After that, *Req* drops, resetting *LB* and in turns *Ack* to low. At this point the circuit returns to its initial state, ready for the next conversion.

3. Constructing a STG from the Specification

A STG describing the operation of the self-timed controller is shown in Figure 2. In this STG, vertices represent control events corresponding directly to rising and falling transitions of signals in the controller; directed arcs between transitions are timing precedence constraints. Transitions of input signals to the circuit are underlined to distinguish them from those of *non-input* signals. Precendence constraints of the former are given by the specification and are assumed to be satisfied by the outside world, whereas precedence constraints of the latter are generated and satisfied by the circuit obtained from the STG. The dashed arcs in this figure are not derived from the behavior specification of the circuit, but are extra constraints to make the STG *persistent*, as will be explained later. The meaning of the transitions are also described in this figure. The notation $a+ \rightarrow b-$ indicates that the rising of signal a has to precede the falling of b, and that the transition $b-$ *is directly caused* by $a+$. The notation $a+ \rightarrow_p b-$ indicates that the occurrence of $a+$ precedes that of $b-$ but it may or may not *directly* precede $b-$. The And-fork relation $a\mathcal{R}_A(b,c)$ means that $a \rightarrow b$ *and* $a \rightarrow c$, and that the occurrence of a will cause the concurrent occurrences of b and c. The And-join relation $(a,b)\mathcal{R}_A c$ means that $a \rightarrow c$ and $b \rightarrow c$, and that c occurs only after both a and b have occurred.

There are two fundamental properties of STG concerning the synthesis of hazard-free and deadlock-free control circuits, those of *liveness* and *persistency.* A STG satisfying these properties is well-formed and possesses a hazard-free and deadlock-free realization. If a STG is strongly connected and for every transition of signal t, there exist at least a *directed path* from $t+$ to $t-$ and at least one from $t-$ to $t+$, then the corresponding circuit realization is *live*, i.e., free from deadlocks. The second property is *persistency*, which states that if a signal is *enabled* in some state of the system, only a transition of that signal can bring the system to another state in which it is no longer enabled. In terms of STG notations, Fig. 3a illustrates an

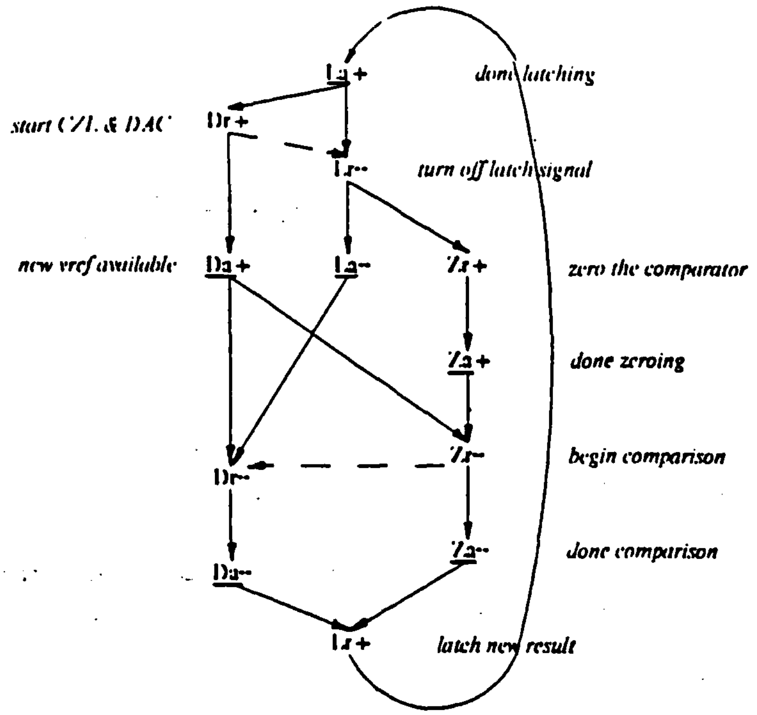


Figure 2: The STG description of the self-timed controller

example of a violation of persistency: transition $a+$ causes transition $b+$, and also, transition $a-$ is concurrent with $b+$. Thus, while b is going high, transition $a-$ may occur and remove the enabling condition of transition $b+$, resulting in a hazard of signal b. Fig. 3b shows that an additional constraint $b+ \rightarrow_p a-$ is required to satisfy the persistency property.

The STG in Fig. 2 is a strongly connected digraph which satisfies the liveness property, therefore it corresponding circuit is live. The transitions D_r-, D_a- and L_a- are merely reset transitions of the reset signaling protocol. Two constraints in this graph deserve careful attention: the constraint $D_a+ \rightarrow Z_r-$ is required because a comparison is not allowed to begin until after a new value of v_{ref} is available; the other constraint $L_r- \rightarrow Z_r+$ makes sure that the gating signal of the latch is turned off before the comparator changes its output value.

4. Making the STG persistent

In general, a STG constructed from the behavior specification of a control circuit alone may not be realizable. In order to have a deadlock-free and

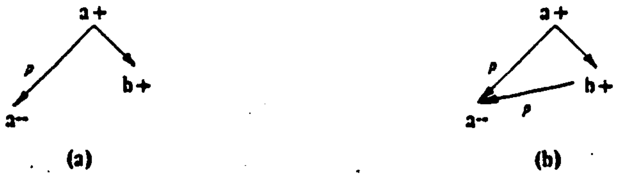


Figure 3: (a) A violation of persistency, (b) elimination of the violation

hazard-free realization, it must satisfy two fundamental properties stated above. In the original specification in Figure 2 (without the dashed arcs), one can immediately detect two cases of violation of persistency. The fork $L_a + \mathcal{R}_A(D_r+, L_r-)$ indicates that L_a+ causes D_r+, but transition L_a- which directly follows L_r- is concurrent with D_r+. Thus, while D_r+ is occurring, L_a- may occur and remove the enabling condition of transition D_r+. The dashed arc $D_r+ \rightarrow L_r-$ eliminates this problem by adding the constraint $D_r+ \rightarrow_p L_a-$ to the graph. In this case a direct constraint $D_r+ \rightarrow L_a-$ is not allowed because L_a- is a transition of an input node of the corresponding circuit; constraints to input transitions are given by the specification and there can be exactly one arc incident to an input transition. Similarly, the violation at the fork $D_a + \mathcal{R}_A(D_r-, Z_r-)$ is eliminated by adding the dashed arc $Z_r- \rightarrow D_r-$ to the graph. After adding these two constraints to eliminate violations of persistency, one can further eliminate constraints $L_a+ \rightarrow L_r-$ and $D_a+ \rightarrow D_r-$ as they become redundant. At this stage, an STG as shown in Figure 4a is obtained.

It turns out that this modified graph is still not implementable due to the lack of internal state information. This is a subtle case of *violation of persistency* which will be discussed later. An internal state signal called x is introduced and the transitions $x+, x-$ are added as shown in Figure 4b. In order to understand how this node is introduced into the STG specification, the synthesis process from STG needs be discussed before this issue can be explored (Section 6.)

5. Synthesis of Circuit from STG

In the circuit realization of this graph, nodes L_a, D_a, Z_a are input nodes to the circuit, other nodes are non-input ones whose logic equations have to be

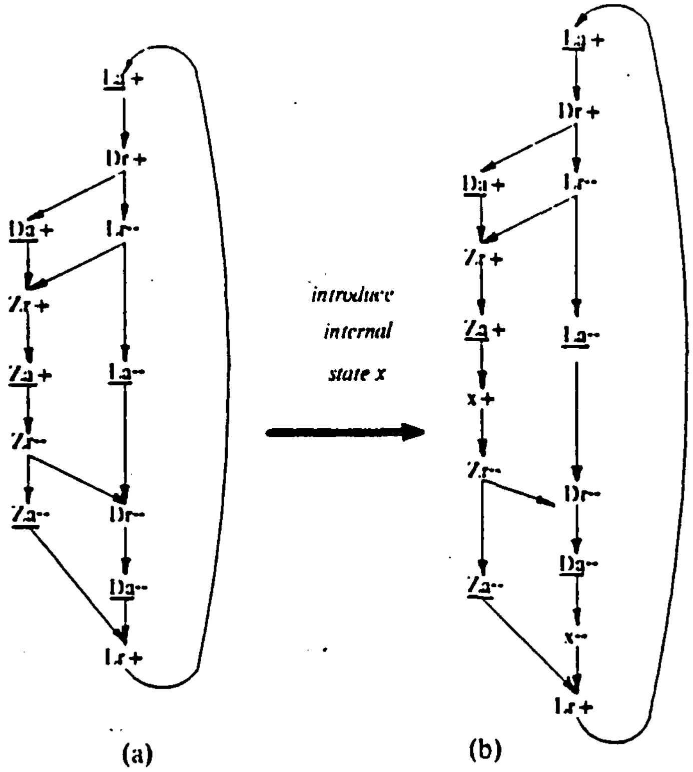
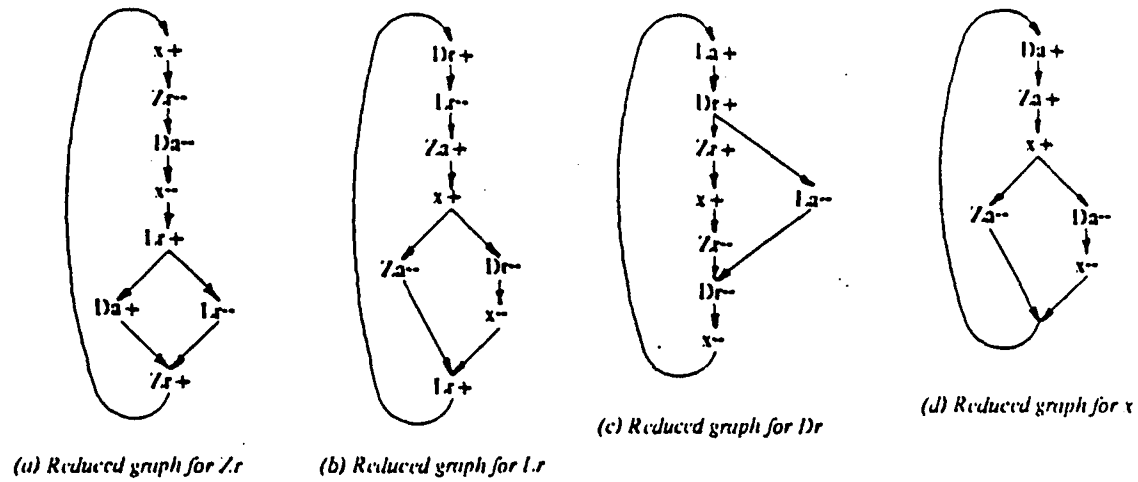


Figure 4: (a) The STG with two violations of persistency eliminated, and (b) the STG with the addition of the internal state x.

determined. The set of non-input nodes is $\{D_r, L_r, Z_r, x\}$. First, the STG in Fig. 4b is decomposed into number of *reduced graphs*, as described below. For each non-input node t_i in set $\{D_r, L_r, Z_r, x\}$, we find its *input set* $I(t_i)$ defined as the set of nodes whose transitions *directly precede* transition t_i, i.e.,

$$I(t_i) = \{t_j \mid t_j \rightarrow t_i\}.$$

Thus, in the STG in Fig. 4b, $I(L_r) = \{D_r, Z_a, x\}$, $I(Z_r) = \{L_r, D_a, x\}$, $I(x) = \{Z_a, D_a\}$ and $I(D_r) = \{L_a, Z_r, x\}$. The last insput set $I(D_r)$ is a special case because even though node x does not directly precede D_r, it is included in $I(D_r)$ to avoid a violation of persistency, as will be discussed in Section 6. A *reduced graph* $G_R(t_i)$ of node t_i is then obtained by removing all transitions in the STG that do not belong to the set $I(t_i) \cup \{t_i\}$, keeping all precedence constraints intact. The reduced graphs $G_R(D_r)$, $G_R(Z_r)$, $G_R(L_r)$ and $G_R(x)$ are shown in Fig. 5. A reduced graph for a non-input node, e.g. Z_r, contains transitions of nodes in the set $I(Z_r) \cup \{Z_r\}$, and the logic element that realizes node Z_r has one output terminal being Z_r and input terminals being those in the set $I(Z_r)$. This graph contains all the



(a) Reduced graph for Zr

(b) Reduced graph for Lr

(c) Reduced graph for Dr

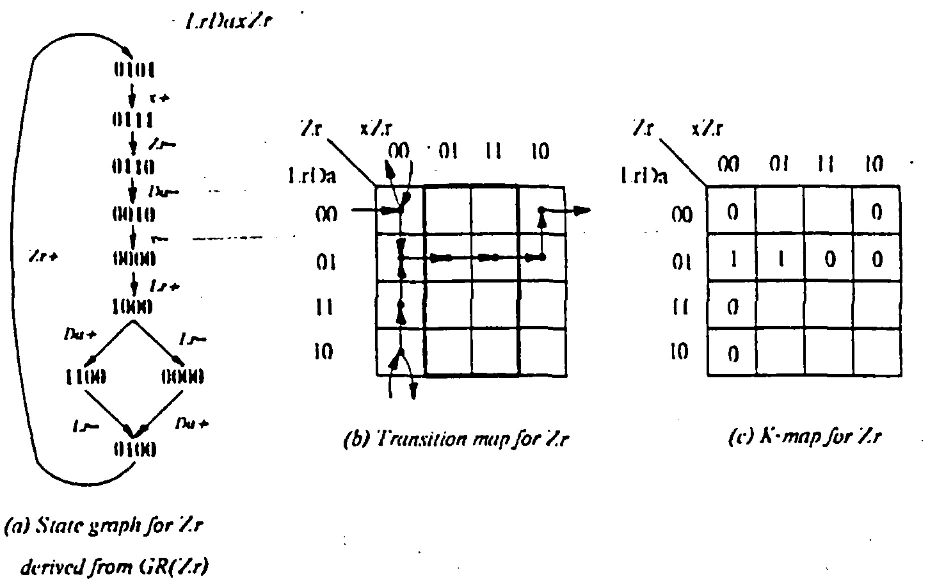
(d) Reduced graph for x

Figure 5: Reduced graphs for non-input nodes in the set $\{Z_r, L_r, D_r, x\}$

information about the timing behavior of logic element Z_r as specified by the precedence relations between its input and output terminals. A logic equation can be determined from this specification as shown next.

The final step in the synthesis process is to derive logic equations from reduced graphs. This step is illustrated for nodes Z_r and x. The equations for D_r and L_r will be given later and the readers can check them using the procedure described.

From the reduced graph $G_R(Z_r)$, one can derive a state graph (Fig. 6a) in which a state is a binary vector representing the state of terminals of logic element Z_r. This state is $L_rD_axZ_r$. The transition from one state to another involves a single variable change, and it corresponds to a *signal-transition* in the reduced graph $G_R(Z_r)$ of Fig. 5a. For example, the state-transition 0101 → 0111 in Fig. 6a is caused by signal transition $x+$ in $G_R(Z_r)$. The concurrent transitions of D_a+ and L_r- is described by a 2-cube of states, containing 2 possible sequence of state changes 1000 → 1100 → 0100 and 1000 → 0000 → 0100. The first state sequence takes places if D_a+ occurs before L_r-, the second takes place if L_r- occurs before D_a+. Since the circuit behaves exactly the same no matter which transition occurs first, it is clear that these two cases also cover the case when L_r- and D_a+ occur at exactly the same time. This is how concurrency can be described in a state-based formulation.



(a) State graph for Zr derived from GR(Zr)

(b) Transition map for Zr

(c) K-map for Zr

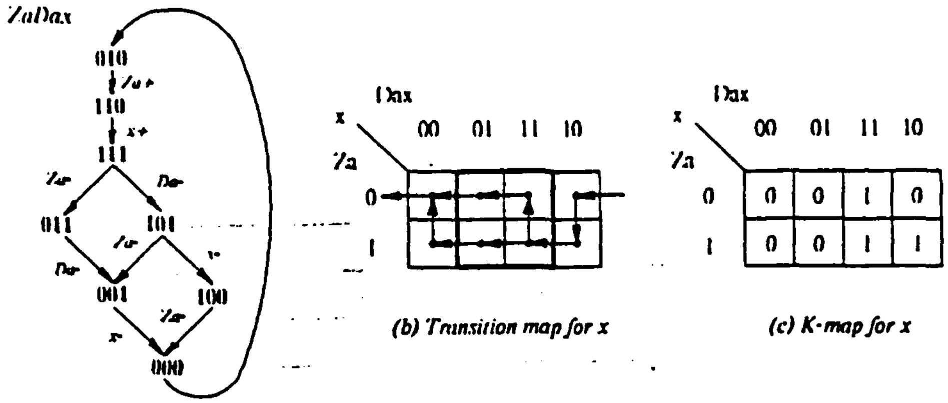
LrDa \ xZr	00	01	11	10
00	0			0
01	1	1	0	0
11	0			
10	0			

Figure 6: Steps in the transformation from a reduced graph to the logic equation for node Z_r.

An *output-conflict* exists if a state has at least two next-states in which values of the output variable are different. In this state graph, state 1000 has two next-states *1100* and *0000*, and they both contain $Z_r = 0$, thus there is no output-conflict. This state graph can be programmed into a type of K-map called *transition map* shown in Fig. 6b. Each entry in this map corresponds to a binary representation of state $L_rD_axZ_r$; arcs between entries are *walks* between adjacent neighbors and they are state transitions given by the state graph. In order to transform a transition-map into a K-map, each entry is replaced by its next-state value of Z_r. For example, in state 0111, the next-state value of Z_r is 0, thus this entry in the transition map is replaced by a 0. If there are more than one next-state and their values of Z_r are different, i.e., if there is an output-conflict, it may not be possible to determine the value of that entry in the K-map. The logic equation of Z_r can be found from this K-map to be

$$Z_r = \bar{L}_rD_a\bar{x}.$$

The derivation of the logic equation for node x is more interesting because this logic element not only has state information but also output-conflicts.



(b) Transition map for x

Za \ Dax	00	01	11	10
0	0	0	1	0
1	0	0	1	1

(c) K-map for x

(a) State graph for x derived from GR(x)

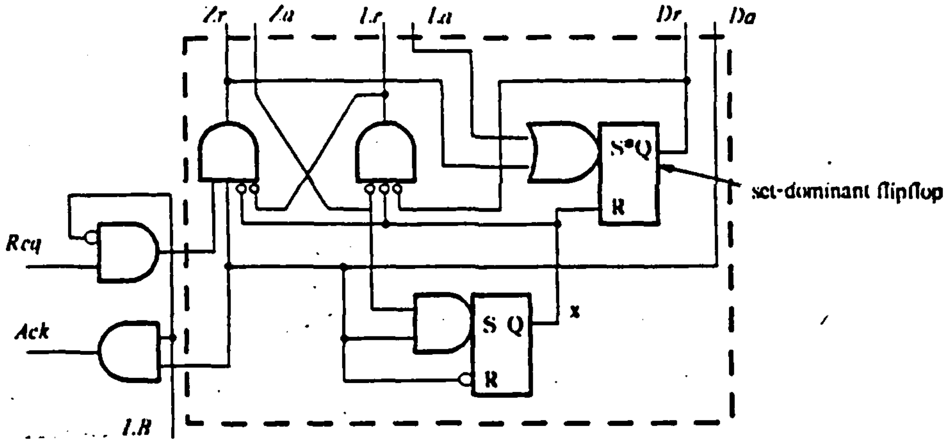
Figure 7: Steps in the transformation from a reduced graph to the logic equation for node x.

The state diagram derived from the reduced graph $G_R(x)$ is shown in Fig. 7a. The concurrent transitions of Z_a- and the *chain* $D_a- \rightarrow x-$ are described by a structure consisting of two 2-cubes, with three possible allowed state sequences $\{(111,011,001,000),(111,101,001,000),(111,101,100,000)\}$. The *output conflict* exists in state $Z_aD_ax = 101$, as x are different in the next-states 001 and 100. However, this output-conflict is caused by concurrent transitions of an input signal Z_a and the output x. In determining the value of x for the K-map in state 101, transition due to $x-$ leading to state 100 must be chosen over transition 101 → 001 because the latter is caused by input transition Z_a-. The state graph in Fig. 7a shows that regardless of whether one is in state 101 or 001, transition $x-$ will always occur next and the circuit behaves exactly the same. The transition map in Fig. 7b doesnot contain the transition 101 → 001. The K-map derived from this state graph is shown in Fig. 7c, the logic equation is found to be

$$x = Z_aD_a + xD_a.$$

This equation has the general form $x = S + x\overline{R}$ with $S = Z_aD_a$ and $R = \overline{D}_a$, its implementation is a set-reset flipflop whose output is x, the set and reset inputs are Z_aD_a and $\overline{D}_a$, respectively. Note that in this implementation, it is required that $S.R = 0$ at all time.

Similarly, the same procedure can be applied to other reduced graphs



Controller's initial state:

Zr=0 Za=0 Lr=0 La=0 Dr=1 Da=1 x=0 LB=0

Figure 8: The final circuit realization of the self-timed controller.

to obtain the logic equation for L_r and D_r. They are $L_r = \overline{D}_r\overline{x}\overline{Z}_a$ and $D_r = Z_r + L_a + D_r\overline{x}$. The equation for D_r can be rewritten as $D_r = S + D_r\overline{R}$ with $S = Z_r + L_a$ and $R = x$, and it is implemented as a set-reset flipflop. The reduced graph of D_r in Fig. 5c shows that there is a time period during which both Z_r and x are high, causing both the set and reset inputs of the D_r flipflop to be active. However, it also indicates that output D_r is not to be reset until after both Z_r and L_a go low, and therefore, until after the set input goes low. The implementation of this flipflop is a *set-dominant* one, as indicated by an asterisk in Fig 8. On the other hand, one can choose to implement D_r directly from the equation given above instead of a set-reset flipflop and not worrying about this particular detail.

Finally, by putting all these elements together, one obtains the control circuit for the A/D converter as shown in the dashed box of Fig. 8. The readers should be able to verify that the self-timed control circuit shown is speed-independent, i.e., it operates correctly with any combination of delays of logic gates, assuming that the internal feedback delays of flipflops are negligible compared to other loop delays in the control circuit.

6. Introducing Internal Node x

The synthesis procedure of the controller from a *well-formed* STG specification have been described. We now return to the original STG specification in Fig. 4a and explain how the internal node x is inserted into this graph to give the well-formed specification in Fig. 4b. The STG in Fig. 4a has non-input nodes D_r, L_r, Z_r, and their input sets are

$$\begin{aligned} I(D_r) &= \{L_a, Z_r\} \\ I(L_r) &= \{D_r, D_a, Z_a\} \\ I(Z_r) &= \{L_r, D_a, Z_a\}. \end{aligned}$$

Their reduced graphs derived from the above STG are shown in Fig. 9. The state graphs of $G_R(D_r)$ and $G_R(L_r)$ are Figs. 9d and e, respectively. The reduced graph of D_r (Fig. 9a) shows that there is an instance of consecutive transitions of node Z_r which directly precedes the output transition D_r ($Z_r+ \rightarrow Z_r- \rightarrow D_r-$). Generally, if an input signal to a logic element changes twice without any intervening transition which alters the state of the system, a hazard may result at the output of that logic element. The state graph of $G_R(D_r)$ (Fig. 9d) in which each state is of form $L_aZ_rD_r$ contains two instances of each of states 101 and 001. In state 101, the output D_r is not enabled and D_r does not make a transition from this state. However, state 001 has an output-conflict in the output variable D_r because the next-states of 001 are 011 and 000. This output-conflict causes hazard because whenever the logic element D_r gets into the upper state 001 both transitions D_r- and Z_r+ are enabled. If Z_r+ occurs first, the state-sequence 001 → 011 → 001 → 000 will take place and logic element D_r behaves correctly. However, if the delay of logic element D_r is smaller than that of Z_r, then transition D_r- will occur first in state 001, even before the occurrence of the chain $Z_r+ \rightarrow Z_r-$. This is a malfunction. In the case when both Z_r and D_r have approximately the same delay, both transitions D_r- and Z_r+ can occur simultaneously, resulting in a hazard at node D_r.

The reduced graph of L_r (Fig. 9b) contains a similar chain of transitions $Z_a+ \rightarrow Z_a-$ directly preceding an output transition L_r+. However, this case has no hazard because after transition Z_a+ occurs, transitions Z_a- and D_r- will occur concurrently. The occurrence of D_r- changes the state of the logic element before Z_a returns to low. Its state graph (Fig. 9e) shows that output transition L_r+ is enabled in the unique state0000, and there is no output-conflict in this case.

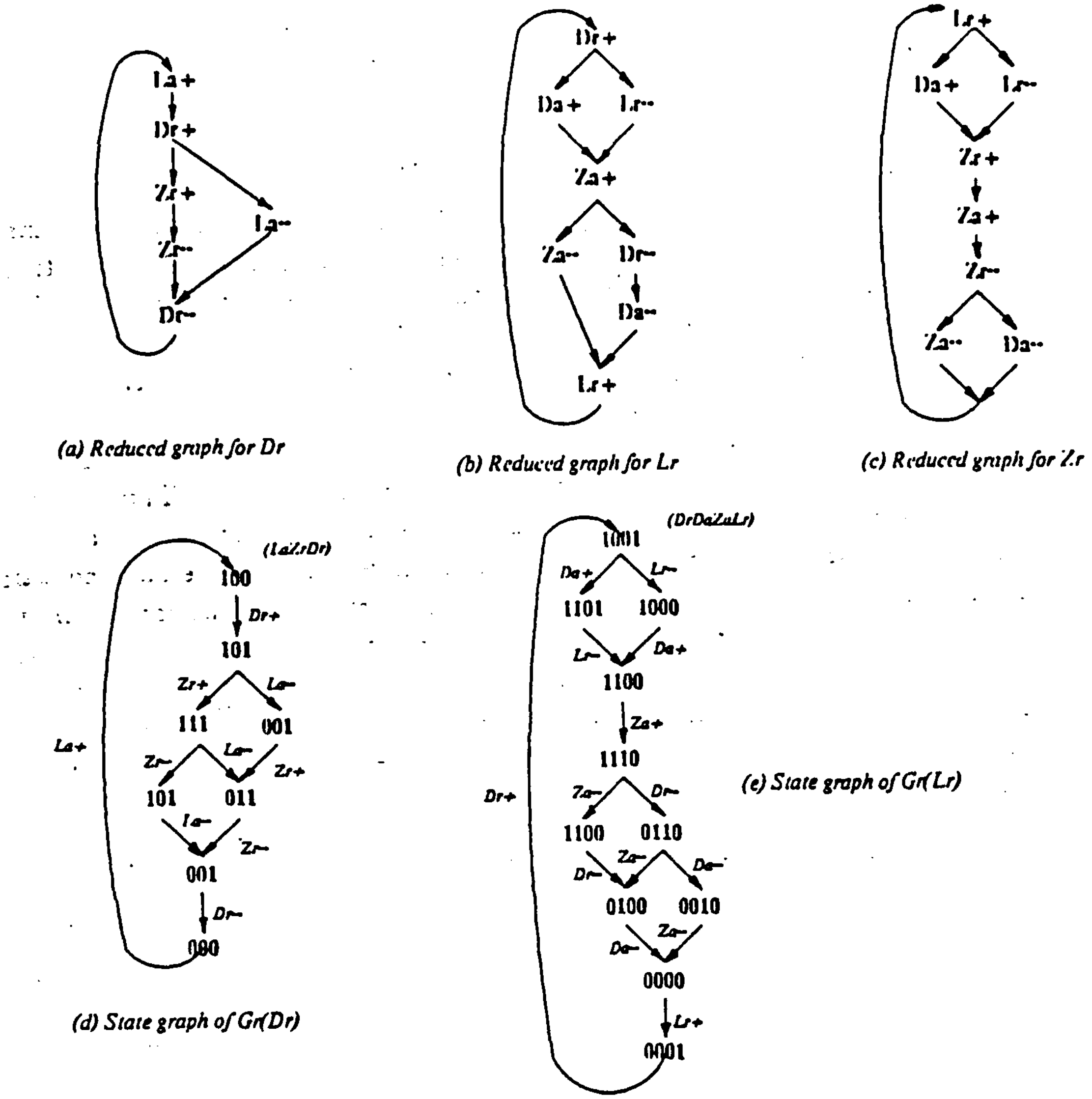
Thus, the hazard at node D_r is caused by the lack of internal state information to discern two instances of state 001. This problem shows up in the reduced graph $G_R(D_r)$ as a pair of consecutive transitions of the same node Z_r. In order to remove this problem, an extra transition such as $x+$ is inserted between these two transitions. Now, transition $x-$ must be inserted into the graph to preserve its liveness. The reduced graph $G_R(D_r)$ shows that $x-$ cannot be inserted (i) between the pair (D_r+, Z_r+) or (Z_r-, D_r-) because this only produces the same problem but with *two* pairs of consecutive transitions of the same nodes; (ii) into the path $(D_r+, La-, D_r-)$ because $x+, x-$ become concurrent and this would violate both the liveness and persistency conditions. Thus $x-$ has to be inserted into the path (D_r-, L_a+, D_r+). Considering the original STG of Fig. 4a, this means that $x-$ must be inserted into the path contain transitions $(D_r-, D_a-, L_r+, L_a+, D_r+)$. Furthermore, transitions of input nodes D_a, L_a and Z_a can have only one incident arcs coming from transitions of their corresponding *request* signals D_r, L_r and Z_r, respectively. Thus $x-$ can be inserted between (D_a-, L_r+) as shown in Fig. 4b. In this final well-formed specification, transition $x+$ does not directly precede transitions of node D_r; however, as explained earlier, it is used as an input to logic element D_r to eliminate hazards at node D_r.

Finally, note that transition $x-$ can also be inserted between (L_a+, D_r+) in the STG in Fig. 4a. This results in another well-formed graph, from which a different implementation can be obtained using the synthesis steps described above. This fact indicates that the implementation is sensitive to the particular form of the STG. This is understandable as the state graphs extracted from STGs are unique state-based representations of the behavior of a circuit.

REFERENCES

[1] Chu, T.-A. "Signal Transition Graphs and the Modeling of Self-timed Circuits." Ph.D. thesis, MIT Dept. of EECS, expected December 1985.

[2] Glasser, L.A. "Synchronizer Failure in A/D Converter." MIT VLSI Memo, Dept. of EECS, July 1985.



(a) Reduced graph for Dr

(b) Reduced graph for Lr

(c) Reduced graph for Zr

(d) State graph of Gr(Dr)

(e) State graph of Gr(Lr)

Figure 9: (a)-(c) Reduced graphs of nodes D_r, L_r and Z_r derived from the STG in Fig. 4a. (d) State graph of $G_R(D_r)$. (e) State graph of $G_R(L_r)$.

DEPARTMENT OF ELECTRICAL ENGINEERING AND COMPUTER SCIENCE

MASSACHUSETTS INSTITUTE OF TECHNOLOGY
CAMBRIDGE, MASSACHUSETTS 02139

VLSI Memo No. 85-276 October 1985

Synchronizer Failure in A/D Converters*

Lance A. Glasser**

ABSTRACT

Analog-to-digital converters that must produce a valid output in a specified period of time are subject to synchronizer failure. Three types of A/D converters are examined: flash converters, clocked successive approximation converters, and self-timed successive approximation converters. Lower bounds on their worst-case conversion time as a function of the fault probability are derived.

*This research was supported by the Defense Advanced Research Projects Agency under contract number N00014-80-C-0622.

**Department of Electrical Engineering and Computer Science and Research Laboratory of Electronics, MIT, Room 36-880, Cambridge, MA 02139, (617) 253-4677.

Synchronizer Failure in A/D Converters

Lance A. Glasser*
Department of Electrical Engineering and Computer Science
and the
Research Laboratory of Electronics
Massachusetts Institute of Technology, room 36–880
Cambridge, Massachusetts 02139
(617) 253–4677

Abstract: Analog-to-digital converters that must produce a valid output in a specified period of time are subject to synchronizer failure. Three types of A/D converters are examined: flash converters, clocked successive approximation converters, and self-timed successive approximation converters. Lower bounds on their worst-case conversion time as a function of the fault probability are derived.

1. Introduction

The finite gain-bandwidth product of the comparators used in analog-to-digital (A/D) converters gives rise to fundamental limits on the speed of various converter architectures. While the average delay of an A/D converter can be made quite fast, the worst-case conversion time of any A/D converter is unbounded [1]. This is because there is always *some probability* that, if the output of an A/D converter is used in a synchronous digital computer system, a fatal synchronizer fault can occur. Our objective is to express a lower bound on the "worst-case" A/D conversion time as a function of this fault probability.

The synchronizer problem in digital systems has been studied for many years [2–14]. Briefly stated, the problem occurs when an asynchronous signal is gated into a clocked system. The clocked system must decide, in a specified period of time, the state of the asynchronous input. The circuit that must make this decision is called a synchronizer.

The synchronizer problem occurs because the synchronizer circuit can take an arbitrarily long time to decide, in marginal cases, whether the input is above or below a given reference standard. In most physical circuits this phenomenon can be attributed to the gain-bandwidth tradeoff. Since, at any given time, the input can be arbitrarily close to the comparator reference, the amplification needed to turn this difference into a full logic swing can be arbitrarily large. For a circuit with a fixed gain-bandwidth product, the delay incurred in achieving this enormous amplification can grow arbitrarily large. If the circuit is required to produce a valid output by a certain time, then there is a finite probability $\mathcal{P}$ that the output will be invalid (not a logic one or a logic zero) at that time. $\mathcal{P}$ decreases exponentially with the time that synchronizer is given to make its decision. Many ingenious techniques, such as adding hysteresis or noise, have been tried to circumvent the synchronizer problem, but the dilemma appears fundamental. The standard solution has become to just wait a sufficient period of time that $\mathcal{P}$ is acceptably small.

* This research was supported by the Defense Advanced Research Projects Agency under contract number N00014-80-C-0622.

If one could build an A/D converter that could be guaranteed to always produce a valid digital output in a bounded amount of time, then one could use this converter to build a fault-free synchronizer. Indeed, the typical synchronizer is nothing more than a one bit A/D converter. This is reason to suspect that the perfect A/D converter does not exist. Tognoni has examined this problem experimentally and observed metastable problems in a commercial A/D converter [15]. In the remainder of this paper we will examine synchronizer problems in A/D converters theoretically, deriving lower bounds on the latency of the A/D conversion as a function of the required reliability.

2. The Model

There are many circuit modules in a typical A/D converter but all A/D converters make use of one or more comparators. We focus our attention, in this paper, on the speed of the comparator.

There are basically two types of comparators. The simplest takes the form of a high gain amplifier; the output voltage of which limits at zero and the positive power supply voltage. We normalize the power supply voltage to 1. If the gain of the high gain amplifier is A, then the output of the comparator is undefined (not a logic 0 or 1) if $|v_{in} - v_{ref}| < 1/(2A)$. v_{in} is the input voltage and v_{ref} is the comparator reference voltage. Thus, there is a finite input voltage range over which the output takes an infinitely long time to settle and if we compute the average comparator response time over all input voltages (all voltages are assumed equally likely) then the average response time diverges to infinity. The delay of a well-designed multistage amplifier takes the form $T_{delay} = \tau \ln A$, where τ is a constant of the technology. (A simple way to view this is to cascade two high-gain amplifiers. The gains multiply and the delays add.)

A bistable (or regenerative) comparator circuit has better average delay properties. Figure 1 illustrates a simple bistable CMOS comparator circuit clocked on ϕ. The delay T_{delay} of this circuit is roughly

$$T_{\mathbf{delay}} = -\tau \ln |v_{\mathbf{in}} - v_{\mathbf{ref}}|, \tag{1}$$

where v_{in} and v_{ref} are normalized to the power supply voltage and τ is roughly equal to the inverse of the gain-bandwidth product for the individual inverters (transistors M_1 through M_4). We normalize the delay to τ. Rewriting (1) in normalized form, we have

$$T_{\mathbf{delay}} = -\ln |\Delta|, \tag{2}$$

where we have defined

$$\Delta \equiv v_{\mathbf{in}} - v_{\mathbf{ref}}. \tag{3}$$

The delay of this amplifier diverges only logarithmically with Δ and therefore has a finite average delay. We will confine our study to this second type of comparator.

The average speed of the comparator is a function of the distance between adjacent bits. For an N bit converter, the (normalized) voltage difference v_{bit} between adjacent bits is

$$v_{bit} = 2^{-N}. \tag{4}$$

The binary fractions range from 0 to $1 - 2^{-N}$.

It is convenient to partition the input voltage into a digital part plus an analog residue. We re-express v_{in} as $v_{\text{in}} = V + v$ where $-v_{\text{bit}} < 2v \leq v_{\text{bit}}$ and V is the ideal voltage representation of an exact N-bit binary fraction, such as the four-bit numbers 0.1011_2 or 0.0001_2. The conversion time is always a function of v and may, or may not, be a function of V, depending on the converter architecture.

In the remainder of this paper we will investigate two types of delay. The average delay T_{avg} is defined as the average of the conversion delays which would be observed if one averaged the times required to perform a large number of conversions, where we have assumed that all input voltages v_{in} are equally likely. We define the worst-case delay $T_{\text{worst-case}}$ in terms of the fault probability $\mathcal{P}$. If we require $\mathcal{P} = 0$ then $T_{\text{worst-case}} \rightarrow \infty$. What we mean by the worst-case conversion time is that if we observe a very large number of conversions M of randomly selected input voltages (uniform distribution) and we throw out the slowest $\mathcal{P}M$ of these conversions as causing faults, then the worst-case conversion time $T_{\text{worst-case}}$ is the slowest conversion time in the remaining set of $(1 - \mathcal{P})M$ better conversions.

3. Flash Converter

For the flash converter, there exists some comparator for which the input voltage is within $\pm v_{\text{bit}}/2$ of the reference voltage. For this comparator we may write $\Delta = v$. Assume that all values of v are equally likely over the interval $(-v_{\text{bit}}/2, v_{\text{bit}}/2]$. The average value of T_{delay} is given by

$$T_{\text{avg}} = -\frac{1}{v_{\text{bit}}} \int_{-v_{\text{bit}}/2}^{v_{\text{bit}}/2} \ln |v| \mathrm{d}v. \tag{5}$$

Solving, we obtain

$$T_{\text{avg}} = 1 + (N + 1) \ln 2 \tag{6}$$

for the flash converter. The minimum average delay increases linearly with N for a given technology.

If $v = 0$, $T_{\text{delay}} \rightarrow \infty$. Given a specified value of T_{delay}, there is some range of v over which the comparator output fails to settle in time. When this happens we say that we have a fault. Let us denote the interval of fault causing voltages by $-\delta < 2v \leq \delta$. The probability of a fault $\mathcal{P}$ is given by

$$\mathcal{P} = \delta / v_{\text{bit}}. \tag{7}$$

We can bound the worst-case delay $T_{\text{worst-case}}$ to be

$$T_{\text{worst-case}} > \ln \frac{1}{v_{\text{bit}} \mathcal{P}}. \tag{8}$$

The interpretation of (8) is that if we perform conversions of M independent input voltages, we expect T_{delay} to be greater than $T_{\text{worst-case}}$ for at least $\mathcal{P}M$ conversions. Solving (8), we obtain

$$T_{\text{worst-case}} > N \ln 2 - \ln \mathcal{P}. \tag{9}$$

The value of $\mathcal{P}$ for which $T_{\text{worst case}} = T_{\text{avg}}$ is $1/2e$. Note that the dependence of the delay on N becomes relatively less important as $\mathcal{P}$ is made smaller.* Figure 2 illustrates (9) for three different values of $\mathcal{P}$.

4. Clocked Successive Approximation Converter

For a clocked successive approximation A/D converter, the average latency is simply N times the clock period. The length of the clock period depends on the fault probability one is willing to accept. The probability of fault in a noiseless successive approximation converter is simply the probability of not successfully completing a conversion on any one clock cycle. Note that this probability is, because the system was assumed ideal and noiseless, just the probability of faulting on the Nth conversion. This was derived for the flash converter. Since there are N conversions, we have

$$T_{\text{worst-case}} > N^2 \ln 2 - N \ln \mathcal{P}. \tag{10}$$

Equation (10) is plotted in Fig. 3.

5. Self-timed Successive Approximation Converter

We can observe from (1) that the comparator delay is a function of the initial voltage difference. As pointed out by H.-S. Lee [16], not all conversion steps in a successive approximation converter can be within $\pm v_{\text{bit}}/2$ of the reference voltage. This means that some conversions will be fast and some slow. Figure 4 illustrates the block diagram of a self-timed successive approximation converter. The asynchronous logic for this figure was designed by Mr. Tam-Anh Chu and is documented in [17]. The comparator is balanced at the beginning of each conversion. The end of each conversion is sensed by a mutual exclusion circuit [18]. (Because our analysis ignores the significant overhead time of balancing the amplifier, sensing the completion, and changing v_{ref}, these bounds are not as tight as the ones in the previous section. Thus comparisons of self-timed and clocked successive approximation converters should be done with care.)

On each conversion k, a binary number B_k is tested. Let $D_k \equiv B_k - V$ be the difference between the present test value B_k and the final binary voltage. D_k can be either positive or negative and is bounded by $|D_k| \leq 2^{-k}$, where $1 \leq k \leq N$. The delay T_k of the kth conversion (considering only the comparator delay contribution) is given by

$$T_k = -\ln|D_k - v|. \tag{11}$$

Summing over all N conversions,

$$T_{\text{delay}} = -\sum_{k=1}^{N} \ln|D_k - v|. \tag{12}$$

* While the average performance of the amplifier type comparator is poor, the worst-case performance is similar to that of the bistable comparator. This is because, for the same level of reliability, they both are required to provide amplification on the order of $1/(\mathcal{P} v_{\text{bit}})$.

The average conversion time $T_{\rm avg}$ is found by averaging (12) over all v and all V. The averaging over v can be done in closed form. Given a particular V,

$$T_{\rm avg}(N,V) = \sum_{k=1}^{N} T_{Dk}, \tag{13}$$

where we have defined the conversion time for one bit T_{Dk} as

$$T_{Dk} \equiv -\frac{1}{v_{\rm bit}} \int_{-v_{\rm bit}/2}^{v_{\rm bit}/2} \ln|D_k - v| dv. \tag{14}$$

Solving(14), we obtain

$$T_{Dk} = \begin{cases} 1 - \left\{ \left(\left| \frac{D_k}{v_{\rm bit}} \right| + \frac{1}{2} \right) \ln \left(|D_k| + \frac{v_{\rm bit}}{2} \right) - \left(\left| \frac{D_k}{v_{\rm bit}} \right| - \frac{1}{2} \right) \ln \left(|D_k| - \frac{v_{\rm bit}}{2} \right) \right\} & \text{for } D_k \neq 1 \\ 1 - \ln \left(\frac{v_{\rm bit}}{2} \right) & \text{for } D_k = 0 \end{cases} \tag{15}$$

D_k is a function of V and N. Averaging (13) over all V we obtain

$$T_{\rm avg}(N) = 2^{-N} \sum_{V=0}^{1-2^{-N}} \sum_{k=1}^{N} T_{Dk}. \tag{16}$$

The worst-case delay is given by

$$T_{\rm worst-case}(N) > \max_{0 \leq V \leq 1} \sum_{k=1}^{N} - \ln|D_k - v_{\rm bit} \mathcal{P}|. \tag{17}$$

A program was written to preform exhaustive analysis, for all V from $N = 1$ to 14, on (16) and (17). For (17), $\mathcal{P}$ was set to 0.1, 10^{-5}, and 10^{-9}. Figure 5 illustrates the worst-case and average behavior of the self-timed successive approximation converter. Table 1 gives the numbers V which caused the worst-case performance in (17). In many cases there are several numbers which cause equivalently slow behavior. The smallest of these is listed. Empirically, numbers of the form ...010101 always caused worst-case behavior. For N in the region 14–20 and $\mathcal{P}$ in the region of 10^{-9} to 10^{-12}, the lower bound on the self-timed successive approximation converter speed was about five times slower than a flash converter constructed in the same technology.

Table 1. Worst-case V versus N

N	V (base 2)
2	0.01
3	0.011
4	0.0101
5	0.01101
6	0.010101
7	0.0110101
8	0.01010101
9	0.011010101
10	0.0101010101
11	0.01101010101
12	0.010101010101
13	0.0110101010101
14	0.01010101010101

6. Conclusions

Three types of converters were studied with respect to average and worst-case delay. In all cases, the worst-case latency was shown to increase with increasing reliability. Note that while latency is a function of $\mathcal{P}$, the throughput need not decrease with decreasing $\mathcal{P}$, providing one is willing to add extra pipeline hardware to give the synchronizers time to settle.

We can divide A/D applications into two classes. Those in which latency is important (such as industrial control applications in which the A/D is in a feedback path with, say, a microprocessor) and those in which throughput is important (such as for image processing). For the first class of applications, one can anticipate that catastrophe avoidance will mandate high reliability requirements. This implies a small $\mathcal{P}$, in conflict with system stability requirements for low latency. For pipelined signal processing applications, additional pipeline stages can easily be added to reduce $\mathcal{P}$ to an acceptable level.

One of the more intriguing results of this study is that the self-timed successive-approximation converter begins to become competitive, in terms of speed, with flash converters for very low $\mathcal{P}$ and large N. This is because, for low $\mathcal{P}$, both the flash and self-timed successive-approximation converters spend most of their time working on the one hard ($v \ll v_{\text{bit}}$) bit that must be assumed to be there.

7. Acknowledgements

I am pleased to acknowledge stimulating and helpful conversations with Prof. Sid Penstone, Prof. Hae-Seung Lee, Dr. Mark Matson, Mr. Tam-Anh Chu, and Mr. Keith Tognoni.

Figures

References

[1] D. Chapiro, "Global-Asynchronous Locally-Synchronous Systems," Ph.D. Thesis, Stanford University, p. 108, October, 1984. (Report No. STAN-CS-1026)

[2] I. Catt, "Time Loss Through Gating of Asynchronous Logic Signal Pulses," *IEEE Trans. on Electronic Computers,* Vol. EC-15, No. 1, pp. 108–111, 1966.

[3] T. Chaney and C. Molner, "Anomalous Behavior of Synchronizer and Arbiter Circuits," *IEEE Trans. on Computers,* Vol. C-22, No. 4, pp. 421–422, 1973.

[4] T. Chaney and F. Rosenberg, "Characterization and Scaling of MOS Flip Flop Performance in Synchronizer Applications," *Proc., Caltech Conference on Very Large Scale Integration*, pp. 357–374, 1979.

[5] G. Couranz, "An Analysis of Binary Circuits Under Marginal Triggering Conditions," Technical Memorandum #15, Computer Systems Lab, Washington University, St. Louis, 1969.

[6] G. Couranz and D. Wann, "Theoretical and Experimental Behavior of Synchronizers in the Metastable Region," *IEEE Trans. on Computers*, Vol. C-24, pp. 604–616, 1975.

[7] S. Flannagan, "Synchronization Reliability in CMOS Technology," *IEEE J. of Solid-State Circuits*, Vol. SC-20, No. 4, pp. 880–883, 1985.

[8] J. Hohl, W. Larsen and L. Schooley, "Prediction of Error Probabilities for Integrated Digital Synchronizers," *IEEE J. of Solid-State Circuits*, Vol. SC-19, No. 2, pp. 236–244, 1982.

[9] D. Kinniment and J. Woods, "Synchronization and Arbitration Circuits in Digital Systems," *Proc. IEE (England)*, Vol. 123, No. 10, pp. 961–966, 1976.

[10] L. Marino, "The Effect of Asynchronous Inputs on Sequential Network Reliability," *IEEE Trans. on Computers*, Vol. C-26, No. 11, pp. 1082–1190, 1977.

[11] P. Mars, "Study of the Probabilistic Behavior of Regenerative Switching Circuits," *Proc. IEE (England)*, Vol. 115, pp. 662–668, 1968.

[12] M. Pechoucek, "Anomalous Response Times of Input Synchronizers," *IEEE Trans. on Computers*, Vol. C-25, No. 2, pp. 133–139, 1976.

[13] W. Plummer, "Asynchronous Arbiters," *IEEE Trans. on Computers*, Vol. C-21, No. 1, pp. 37–42, 1972.

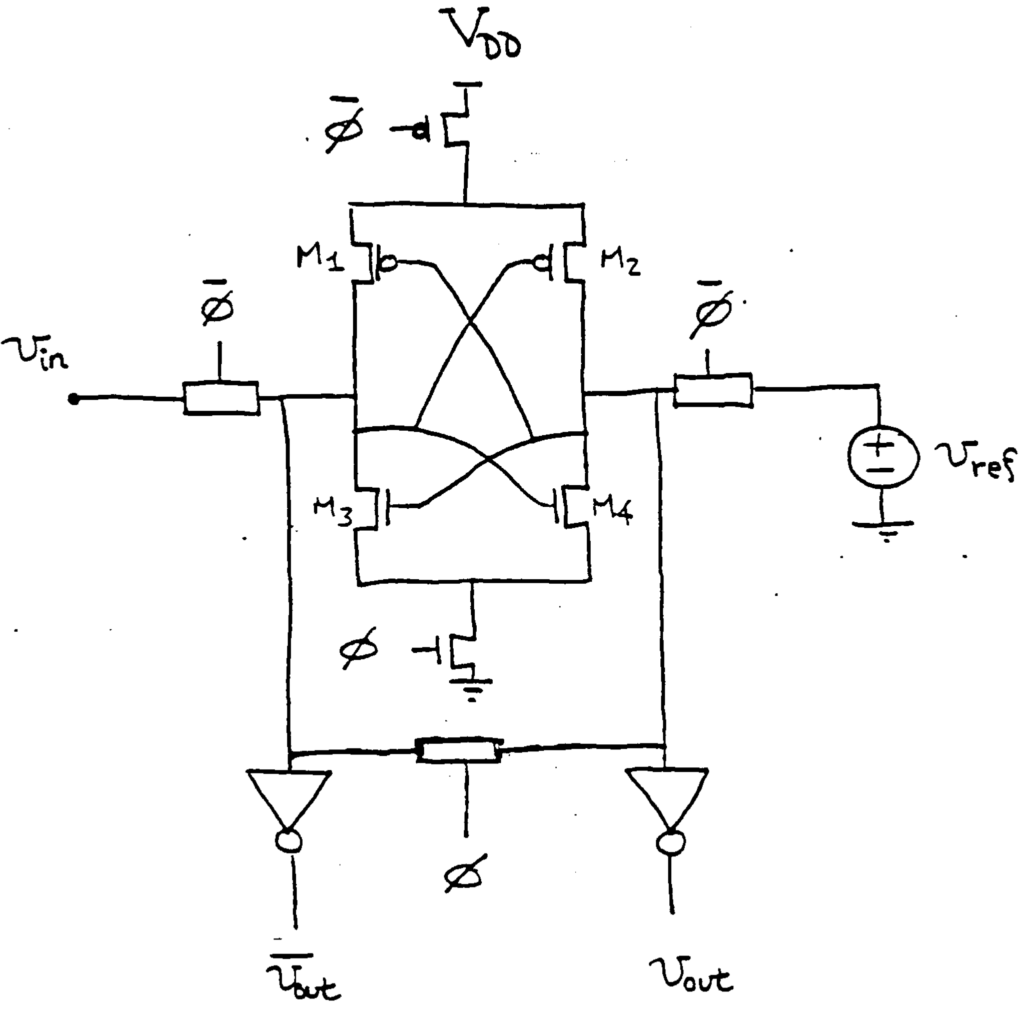
[14] H. Veendrick, "Behavior of Flip-Flops Used as Synchronizers and Prediction of Their Failure Rates," *IEEE J. of Solid-State Circuits,* Vol. SC-15, No. 2, pp. 169–176, 1980.

[15] K. Tognoni, "Metastable Problems in A/D Converters," S.B. Thesis, Massachusetts Institute of Technology, 1985.

[16] H.-S. Lee, private communication.

[17] T.-A. Chu and L. A. Glasser, "Synthesis of a Self-Timed Controller for a Successive-Approximation A/D Converter," VLSI memo, unpublished.

[18] C. Seitz, "Ideas About Arbiters," *LAMBDA*, 1st Q., p. 14, 1980.



V_{DD}
$\bar{\phi}$
M_1
M_2
$\bar{\phi}$
v_{in}
$\bar{\phi}$
v_{ref}
M_3
M_4
ϕ
ϕ
$\bar{v}_{out}$
v_{out}

FIG 1

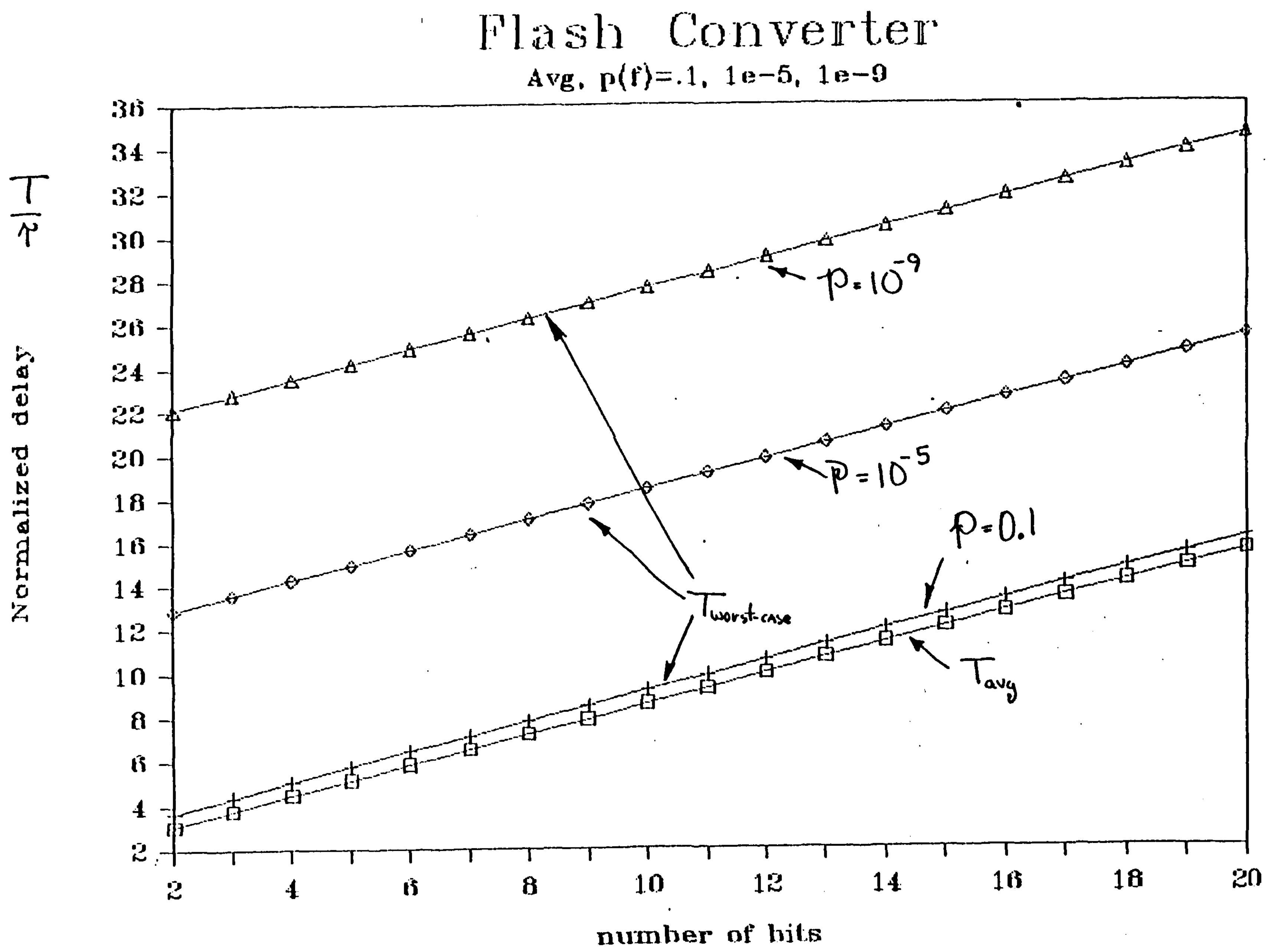

Flash Converter
Avg, p(f)=.1, 1e−5, 1e−9
Normalized delay
$\frac{T}{\tau}$
36
34
32
30
28
26
24
22
20
18
16
14
12
10
8
6
4
2
2
4
6
8
10
12
14
16
18
20
number of bits
$p=10^{-9}$
$p=10^{-5}$
$p=0.1$
$T_{worst-case}$
T_{avg}

FIG 2

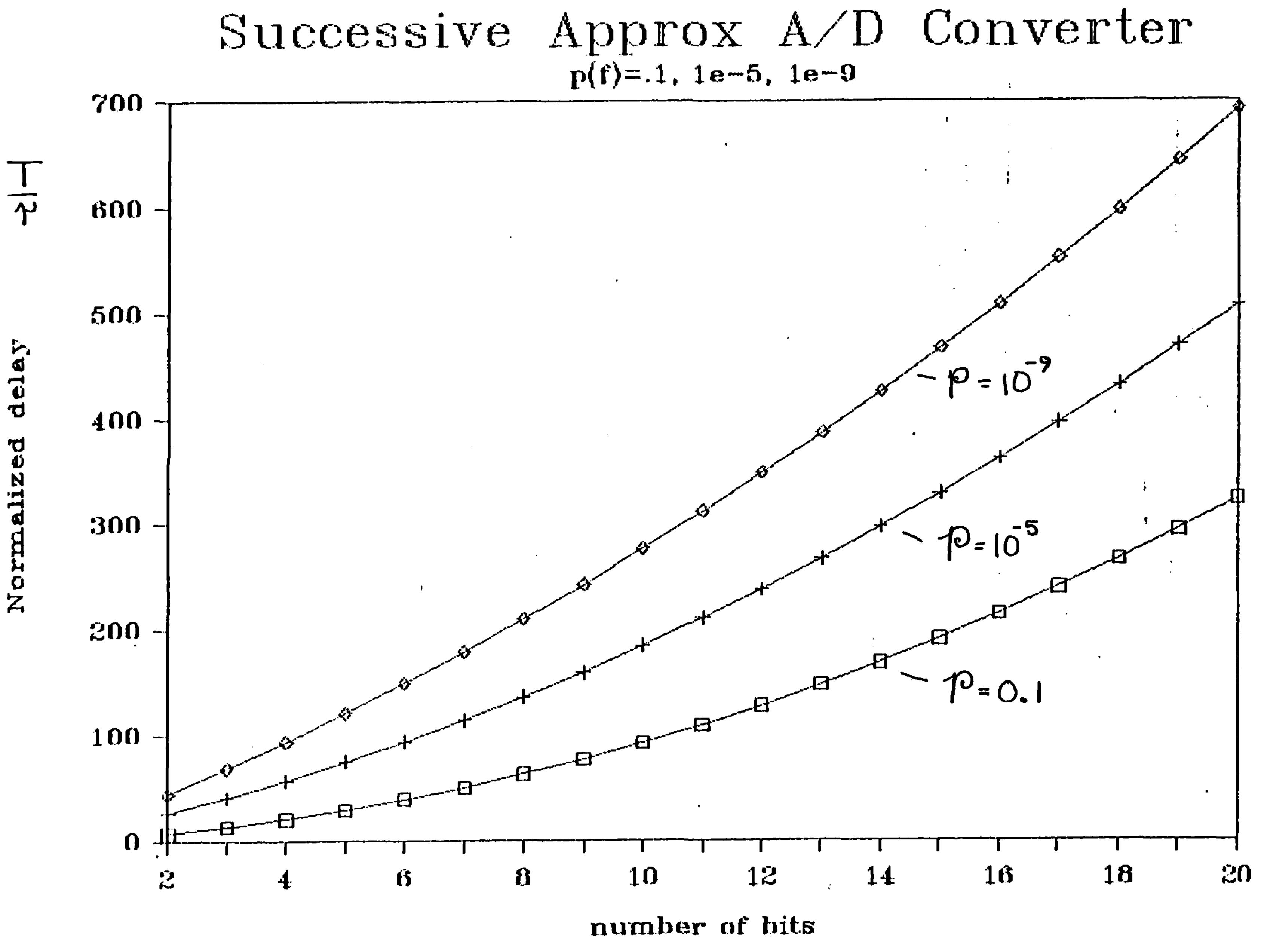

Successive Approx A/D Converter
p(f)=.1, 1e-5, 1e-9
Normalized delay
$\frac{T}{\tau}$
700
600
500
400
300
200
100
0
2
4
6
8
10
12
14
16
18
20
number of bits
$p = 10^{-9}$
$p = 10^{-5}$
$p = 0.1$

FIG 3

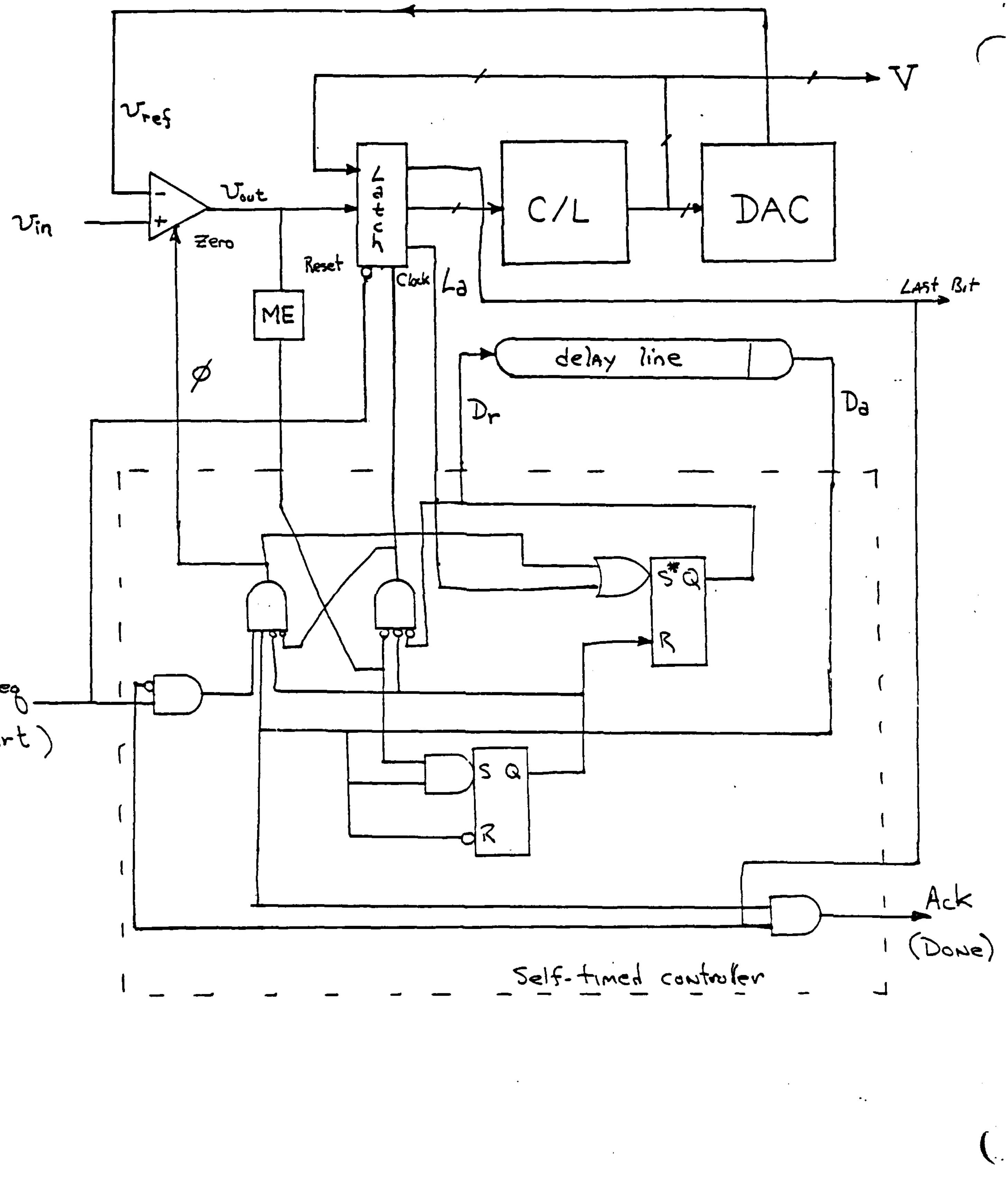
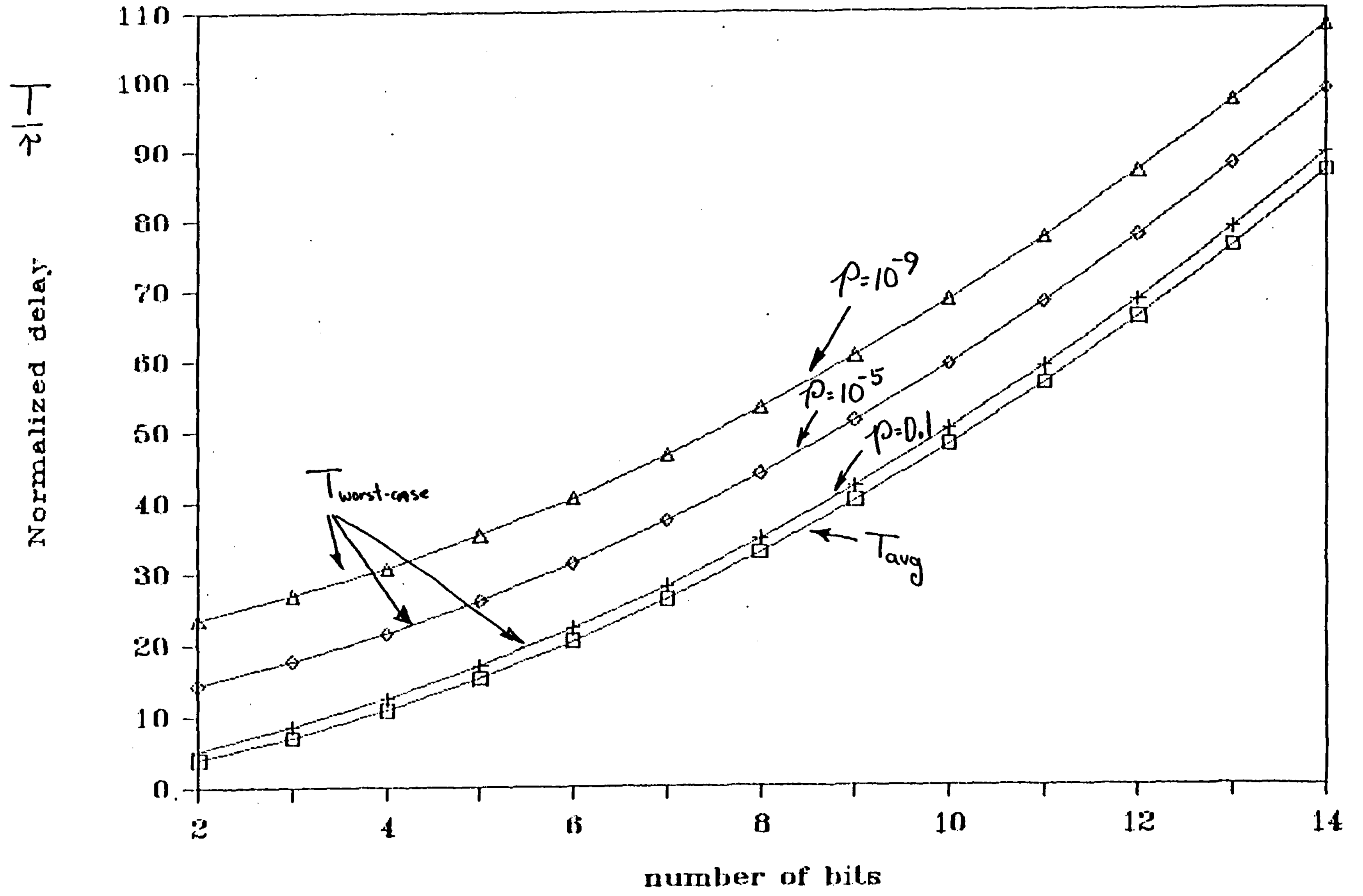

V
v_{ref}
v_{out}
v_{in}
Zero
Latch
C/L
DAC
Reset
Clock
L_a
Last Bit
ME
ϕ
delay line
D_r
D_a
S* Q
R
S Q
R
Req
(Start)
Ack
(Done)
Self-timed controller

FIG 4



Self−timed A/D Converter

avg, p(f)=.1, 1e−5, 1e−9

Fat-Trees: Universal Networks for Hardware-Efficient Supercomputing

CHARLES E. LEISERSON, MEMBER, IEEE

Abstract —This paper presents a new class of universal routing networks called _fat-trees,_ which might be used to interconnect the processors of a general-purpose parallel supercomputer. A fat-tree routing network is parameterized not only in the number of processors, but also in the amount of simultaneous communication it can support. Since communication can be scaled independently from number of processors, substantial hardware can be saved over, for example, hypercube-based networks, for such parallel processing applications as finite-element analysis, but without resorting to a special-purpose architecture.

Of greater interest from a theoretical standpoint, however, is a proof that a fat-tree of a given size is nearly the best routing network of that size. This _universality theorem_ is proved using a three-dimensional VLSI model that incorporates wiring as a direct cost. In this model, hardware size is measured as physical volume. We prove that for any given amount of communications hardware, a fat-tree built from that amount of hardware can simulate every other network built from the same amount of hardware, using only slightly more time (a polylogarithmic factor greater). The basic assumption we make of competing networks is the following. In unit time, at most $O(a)$ bits can enter or leave a closed three-dimensional region with surface area a. (This paper proves the universality result for _off-line_ simulations only.)

Index Terms — Fat-trees, interconnection networks, parallel supercomputing, routing networks, universality, VLSI theory.

I. Introduction

MOST routing networks for parallel processing supercomputers have been analyzed in terms of performance and cost. Performance is typically measured by how long it takes to route permutations, and cost is measured by the number of switching components and wires. This paper presents a new routing network called fat-trees, but analyzes it in a somewhat different model. Specifically, we use a three-dimensional VLSI model in which *pin boundedness* has a direct analog as the bandwidth limitation imposed by the surface of a closed three-dimensional region. Performance is measured by how long it takes to route an arbitrary set of messages, and cost is measured as the volume of a physical implementation of the network. We prove a *universality* theorem which shows that for a given volume of hardware, no network is much better.

Unlike a computer scientist's traditional notion of a tree, fat-trees are more like real trees in that they get thicker further from the leaves. In physical structure, a fat-tree resembles, and is based on, the *tree of meshes* graph due to Leighton [12], [14]. The processors of a fat-tree are located at the leaves of a complete binary tree, and the internal nodes are switches. Going up the fat-tree, the number of wires connecting a node with its father increases, and hence the communication bandwidth increases. The rate of growth influences the size and cost of the hardware as well.

Most networks that have been proposed for parallel processing are based on the Boolean hypercube, but these networks suffer from wirability and packaging problems and require nearly order $n^{3/2}$ physical volume to interconnect n processors. In his influential paper on "ultracomputers" [27], Schwartz demonstrates that many problems can be solved efficiently on a supercomputer-based on a shuffle network [28]. But afterwards, Schwartz comments, "The most problematic aspect of the ultracomputer architecture suggested in the preceding section would appear to be the very large number of intercabinet wires which it implies." Schwartz then goes on to consider a "layered" architecture, which seems easier to build, but which may not have all the nice properties of the original architecture.

On the other hand, there are many applications that do not require the full communication potential of a hypercube-based network. For example, many finite-element problems are planar, and planar graphs have a bisection width of size $O(\sqrt{n})$, as was shown by Lipton and Tarjan [19]. Moreover, any planar interconnection strategy requires only $O(n)$ volume. Thus, a natural implementation of a parallel finite-element algorithm would waste much of the communication bandwidth provided by a hypercube-based routing network.

Fat-trees are a family of general-purpose interconnection strategies which effectively utilize any given amount of hardware resource devoted to communication. This paper proves that for a given physical volume of hardware, no network is much better than a fat-tree. Section II introduces fat-tree architectures and gives the logical structure of one feasible implementation. Section III shows how communication on a fat-tree can be scheduled off-line in a near-optimal fashion. Section IV defines the class of *universal* fat-trees and investigates their hardware cost in a three-dimensional VLSI model. Section V contains several combinatorial theorems concerning the recursive decomposition of an arbitrary routing network, and Section VI uses these results to demonstrate that fat-trees are indeed a class of hardware-efficient universal routing networks. Finally, Section VII offers some remarks about the practicality of fat-trees.

Manuscript received February 1, 1985; revised May 30, 1985. This work was supported in part by the Defense Advanced Research Projects Agency under Contract N00014-80-C-0622. A preliminary version of this paper was presented at the IEEE 1985 International Conference on Parallel Processing, St. Charles, IL, Aug. 1985.

The author is with the Laboratory for Computer Science, Massachusetts Institute of Technology, Cambridge, MA 02139.

0018-9340/85/1000-0892$01.00 © 1985 IEEE

II. Fat-Trees

This section introduces fat-trees as a routing network for parallel computation. The parallel computer based on fat-trees that we present is somewhat arbitrary and is influenced by the various connection machine projects [5], [9], [11] conceived at M.I.T. The computational model is not meant to be exclusive—the results in this paper undoubtedly apply to more general models. Moreover, arbitrary "engineering design decisions," which may not be the best choices from either a practical or a theoretical perspective, have been made in this description of fat-trees. Most of the choices influence the results by only a logarithmic factor, however, and do not affect the overall thrust of the paper—the universality theorem in Section VI.

The intuitive model for parallel computation that we use is a parallel computation engine composed of a set of processors interconnected by a *routing network*. The processors share no common memory, and thus they must communicate through the routing network, using *messages*. The job of the routing network is to see that all messages eventually reach their destinations as quickly as possible.

A fat-tree FT is a routing network based on a complete binary tree. (See Fig. 1.) A set P of n processors is located at the leaves of the fat-tree. Each edge of the underlying tree corresponds to two *channels* of the fat-tree: one from parent to child, the other from child to parent. Each channel consists of a bundle of wires, and the number of wires in a channel c is called its *capacity*, denoted by cap(c). The capacities of channels in the routing network are determined by how much hardware we can afford, a topic to be discussed in Section IV. The channel leaving the root of the tree corresponds to an interface with the external world. Each (internal) node of the fat-tree contains circuitry that switches messages between incoming channels and outgoing channels.

Messages produced by processors are *batched* into *delivery cycles*. During a delivery cycle, a processor may send messages through the network to other processors. Some messages may be lost in the routing network during a delivery cycle. Thus, in general, at the end of the delivery cycle, acknowledgments are sent from the destination processor back to the source processor. Messages that are not delivered must be sent again in subsequent delivery cycles.

The nodes of the fat-tree accomplish most of the switching. In order to understand their function, one must first understand how the routing of messages is accomplished. A *message set* $M \subseteq P \times P$ is a set of *messages* (i, j). If $(i, j) \in M$, then processor i has a message to be sent to processor j. (We omit details concerning the contents of messages and the handling of messages routed to and from the external interface.) Routing in the fat-tree is basically easy since every message has a unique path in the underlying complete binary tree. A message going from processor i to processor j goes up the tree to their least common ancestor and then back down according to the least significant bits of j. Notice that at any node of the fat-tree, there are only two choices for the routing of a message. If it comes into a node from a left subtree, for example, it can only go up or down to the right. Thus, a bit string of length at most 2 lg n is sufficient to represent the destination of any message.[1]

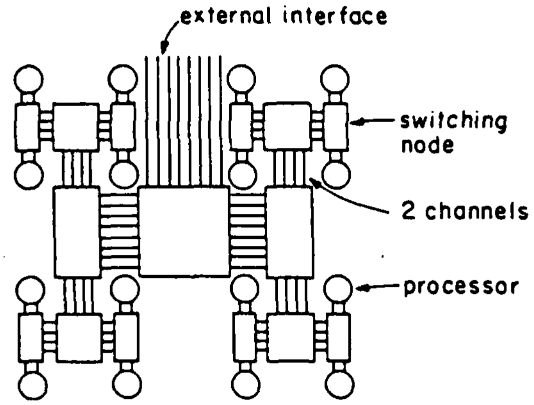


Fig. 1. The organization of a fat-tree. Processors are located at the leaves, and the internal nodes contain concentrator switches. The capacities of channels increase as we go up the tree.

We shall consider communication through the fat-tree network to be synchronous and bit serial. Messages snake through the tree with leading bits of a message establishing a path for the remainder to follow. Since some of the paths through the tree are longer than others, synchronization of the departures and arrivals of messages can be a bit tricky. Buffering of messages by the sending processors is one solution to this problem. (As was mentioned before, there are many other engineering alternatives that lead to the same kinds of theoretical results reported here.) The differing lengths of paths in the fat-tree are actually a major advantage of the network because messages can be routed locally without soaking up the precious bandwidth higher up in the tree, much as telephone communications are routed within an exchange without using more expensive trunk lines.

The messages in the network obey the bit-serial protocol shown in Fig. 2. The first bit is the *M bit*, which tells whether the remaining bits actually contain a message. Next come the *address* bits, which name the destination processor. The final field in the message format is the data themselves. As messages are routed through the network, each node uses the M bit to identify whether a wire carries a message, and it uses the first address bit to make a routing decision. A path is established through the node for a new M bit and the remaining message bits to follow. The address bits are stripped off one by one as the message establishes a path through the network.

A fat-tree node has three input ports, U_I, L_I, and R_I, and three output ports, U_O, L_O, and R_O, connected in the natural way to the wires in the channels. Messages entering input port L_I will go either to output port U_O or to output port R_O. The logic of the switching circuitry in a node consists of three similar portions, shown in Fig. 3. A wire from an input port is fanned out towards the two opposite output ports. The M bit of each wire is then examined to determine whether the wire has a message. On the next clock tick, the first address bit is examined on both branches of the input wire. By ANDing

[1] We use the notation lg n to mean max{1, $\log_2 n$}.

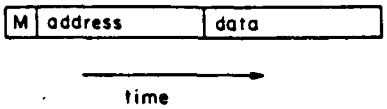


Fig. 2. The format of bit-serial messages. The first bit that a switch sees is the M bit, which indicates whether an input wire actually contains a message. The address bits arrive bit-serially in subsequent time steps, and the message contents are last.

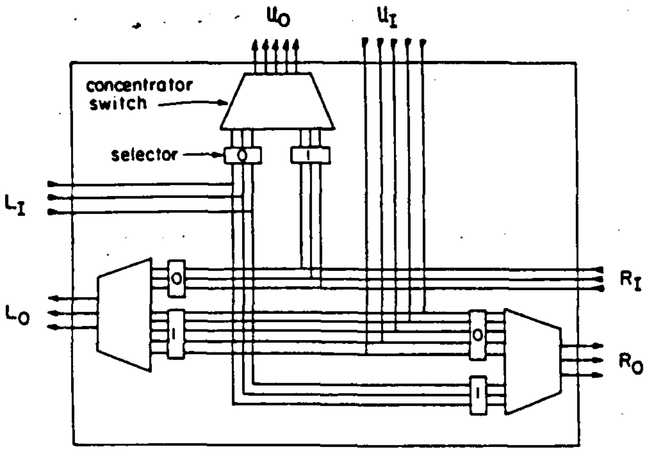


Fig. 3. The internal structure of a fat-tree node. A selector determines which messages are destined for an output port, and then a concentrator switch establishes disjoint electrical paths for as many of those messages as possible.

the M bit with either the address bit or its complement, an M bit is determined for each branch by a *selector*. Next, the messages destined for an output port, which currently occupy many wires, are switched onto fewer wires by a *concentrator switch*.

The job of the concentrator switch is to create electrical paths from those input wires that carry messages to fewer output wires. Obviously, if there are more input messages than output wires, some messages will be lost. In this case we shall say that the output channel is *congested*. We have already mentioned an acknowledgment mechanism that detects when messages are lost due to congestion.

For the time being, we shall assume that the concentrator switch has the following property. If there is no congestion—that is, the number of input messages does not exceed the number of output wires—then no messages are lost. The concentrator switch that we shall present in Section IV is a *partial concentrator* and does not have exactly this property, but it makes little difference to the theoretical results. This circuit has $O(m)$ components if there are a total of m incident wires, and it switches in constant time. Thus, the time required for an entire delivery cycle in a fat-tree of n processors is $O(\lg n)$.

Although we have described the general setting as an *on-line* switching environment, this paper makes the simplifying assumption that the fat-tree nodes contain *off-line* circuitry, in that the switches, although dynamically set, have their settings predetermined by an off-line scheduling algorithm. Naturally, it would be better to dynamically determine the settings themselves in real time, and indeed, it is possible to build such on-line switches, but these results will be reported elsewhere [8]. We have chosen here to prove the weaker off-line results so as to simplify the presentation of the universality theorem in Section VI.

There are several consequences of the off-line assumption that bear mention, however. For example, the results apply to practical situations when the settings of switches can be "compiled," as when simulating a large VLSI design or emulating a fixed-connection network. Also, some of the mechanisms—such as acknowledging the receipt of messages, which is necessary in an on-line environment—can be omitted from the off-line hardware structure, thereby reducing the complexity of the design.

III. Off-Line Scheduling on Fat-Trees

The concentrator switches in the nodes of a fat-tree routing network guarantee that no messages are lost unless there is congestion. This section gives an algorithm for scheduling the delivery of an arbitrary set of messages so that all messages will be delivered. We give a simple value, called the *load factor* of a set of messages, which provides a lower bound on how quickly the messages can be delivered. We show that for an arbitrary message set, off-line scheduling can be done optimally to within a logarithmic factor of the number of processors.

Let us be more precise about the off-line scheduling problem. Let FT be a fat-tree on n processors, and let C be the set of channels in FT. For any channel $c \in C$, the capacity cap(c), which is the number of wires in the channel, is also the maximum number of simultaneous messages the channel can support because we are assuming bit-serial communication. Since each message between two processors determines a unique path in the underlying complete binary tree, we can define load(M, c) to be the total number of messages in a message set M that must go through channel c. We call M a *one-cycle* message set if $\text{load}(M, c) \leq \text{cap}(c)$ for all channels $c \in C$. If all capacity constraints are met, a fat-tree with ideal concentrator switches can route every message in one delivery cycle.

A *schedule* of a message set M is a partition of M into one-cycle message sets $M_1, M_2, \cdots, M_d$ where d is the total number of delivery cycles. A simple lower bound on d for an arbitrary message set M is $d \geq \max_c(\text{load}(M, c)/\text{cap}(c))$, which leads to the following definition.

Definition: Let M be a message set, and let $c \in C$ be a channel in a fat-tree. The *load factor* $\lambda(M, c)$ of a channel c due to M is

$$\lambda(M, c) = \frac{\text{load}(M, c)}{\text{cap}(c)},$$

and the *load factor* of the entire fat-tree due to M is

$$\lambda(M) = \max_{c \in C} \lambda(M, c).$$

A message set M is a *one-cycle* message set if $\lambda(M) \leq 1$.

The simple lower bound on the number d of delivery cycles required for any schedule of M can now be reexpressed as $d \geq \lambda(M)$. The next theorem shows that this lower bound can be achieved to within a logarithmic factor of n.

Theorem 1: Let FT be a fat-tree on n processors, and let C be the set of channels in FT. Then for any message set M *with $\lambda(M) > 1$, there is an off-line schedule $M_1, M_2, \cdots, M_d$ such that $d = O(\lambda(M) \lg n)$.*

Proof: The idea is to partition the messages going from left to right through the root of the fat-tree into at most $2\lambda(M)$ one-cycle message sets, to do the same for the messages going from right to left, and then to recursively partition the messages in the two subtrees of the root. Let Q_1 be the subset of M consisting of those messages that must go through the root from left to right. The scheduling algorithm will begin by partitioning Q_1 into two message sets Q_2 and Q_3. It then iteratively refines each Q_k into Q_{2k} and Q_{2k+1}, until each Q_k, $k = r, \cdots, 2r - 1$ is a one-cycle message set for some $r \le 2\lambda(M)$. The r message sets $Q_r, \cdots, Q_{2r-1}$ form the initial sequence of the schedule.

The algorithm similarly partitions the message set consisting of messages going from right to left in the fat-tree and adds them to the schedule. (Each of these message sets can, in fact, be routed at the same time as one of the Q_k.) Finally, the algorithm recursively partitions the messages remaining within the two subtrees of the root. The upper bound of $2\lambda(M)$ one-cycle message sets holds for all messages routed through the root of a subtree. But since all subtrees with roots at the same level can be routed at the same time, the total number of delivery cycles required is at most the height of the fat-tree times the time for one level, which yields $d = O(\lambda(M) \lg n)$.

It remains to show that the message sets can be partitioned effectively. Consider once again the message set Q_1 of messages going left to right through the root of the fat-tree. We now show that each message set Q_k, $k = 1, 2, \cdots, r - 1$, can be partitioned into Q_{2k} and Q_{2k+1} so that for every channel $c \in C$, the messages of Q_k that go through c are split exactly evenly, that is, so that $\text{load}(Q_{2k}, c) \le \lceil (1/2) \text{load}(Q_k, c) \rceil$ and $\text{load}(Q_{2k+1}, c) \le \lceil (1/2) \text{load}(Q_k, c) \rceil$. The partitioning consists of two parts, matching and tracing, and is reminiscent of switch setting in a Benes network [34] and the Eulerian tour routing algorithm from [10].

First, do the matching. Consider each message in Q_k as being a string with two ends: a source end and a destination end. Within each processor, match as many pairs of string ends as possible until at most one message of Q_k is unmatched within each processor. Notice that source ends are matched only with source ends and destination ends only with destination ends because all messages in Q_k go left to right through the root. Then consider two-leaf subtrees. If each of the two leaves has one unmatched string end, match the ends. Continue matching the unmatched string ends in four-leaf subtrees, and so on up the fat-tree. At every level of the fat-tree, at most one string end is unmatched in each of the two subtrees of a node. At the root, at most one string end from each side will be unmatched (when there is an odd number of messages going from left to right through the root).

Now the tracing phase begins. If there is an unmatched string end in the left subtree, start with it. Otherwise, pick a string end arbitrarily from the left subtree. Put the corresponding message into Q_{2k}, and follow the string to the right subtree. Find the mate of the string end on the right side, and put the corresponding message into Q_{2k+1}. Follow this new string back to the left side, find its mate, and put the corresponding message into Q_{2k}. In general, when traversing a string left to right, put the corresponding message into Q_{2k}. When traversing right to left, put the message into Q_{2k+1}. If we discover that a string end has no mate, or that the message corresponding to the mate has already been assigned, we have either found the (one) unmatched string end on the right or completed a cycle. In either event, pick another string end arbitrarily and continue until all messages in Q_k have been assigned either to Q_{2k} or to Q_{2k+1}.

To see that this algorithm evenly splits the messages of Q_k in every channel c, observe that the number of times we enter a subtree of the fat-tree is equal to the number of times we leave, unless we are tracing the one possible string end matched outside the subtree. Since the split is even in every channel, the partitioning of Q_1 into one-cycle message sets $Q_r, \cdots, Q_{2r-1}$ will be achieved when

$$\begin{aligned} r &\le 2 \max_c \frac{\text{load}(Q_1, c)}{\text{cap}(c)} \\ &\le 2 \max_c \frac{\text{load}(M, c)}{\text{cap}(c)} \\ &\le 2\lambda(M), \end{aligned}$$

which completes the proof. ■

For the special case when $\text{cap}(c) > a \lg n$, for some $a > 1$, the logarithmic factor in the upper bound of Theorem 1 can be removed. Thus, under these conditions, the lower bound of the load factor can be met almost exactly.

Corollary 2: Let FT be a fat-tree on n processors, let C be the set of channels in FT, and suppose that there is a constant $a > 1$ such that $\text{cap}(c) \ge a \lg n$ for all $c \in C$. Then for any message set M, there is an off-line schedule $M_1, M_2, \cdots, M_d$ such that $d = O((a/a - 1)\lambda(M))$.

Proof: For each channel $c \in C$, define a set of *fictitious capacities* $\text{cap}'(c) = \text{cap}(c) - \lg n$. The fat-tree with the fictitious capacities has a load factor $\lambda'(M) \le (a/a - 1)\lambda(M)$. Now use the scheduling algorithm of Theorem 1, but during the recursion on lower levels of the tree, rather than using new message sets, simply reuse the $2\lambda'(M)$ message sets produced by partitioning the messages through the root.

The bisections at a given level produce partitions of the set of messages that are equal to within one, and this error can accumulate in a single channel as we go down the tree. The largest value of the error can be as much as $\lg n$, but the actual capacities are never exceeded, and so each of the $2\lambda'(M)$ message sets will be routable in one delivery cycle. ■

Thus, for example, if the capacities are each at least $2 \lg n$, the number of delivery cycles is not worse than $4\lambda(M)$. (In fact, the divide-and-conquer partitioning of messages can be improved to $2\lambda(M) + o(\lambda(M))$.)

IV. The Hardware Requirements of Fat-Trees

This section investigates the amount of hardware required by a fat-tree. We give a precise description of how the switches in the nodes of a fat-tree might be implemented and determine how much hardware a node requires. We then define the channel capacities of *universal* fat-trees. Finally, we determine the amount of hardware required to build universal fat-trees.

The model for hardware that we use is an extension of Thompson's two-dimensional VLSI model [29] by making the natural extension to three dimensions. In this model, wires occupy volume and have a minimum cross-sectional area. Similar three-dimensional models have been studied by Rosenberg [26] and Leighton and Rosenberg [16].

We first present an implementation of a fat-tree node. As was shown in Fig. 3, most of the switching components are contained in the three concentrator switches. According to the three-dimensional VLSI model, however, we must also be concerned with the amount of wire consumed by the interconnection of the components. We shall show that a fat-tree node with m incident wires can be built with $O(m)$ components in a box whose side lengths are $O(h\sqrt{m})$, $O(h\sqrt{m})$, and $O(\sqrt{m}/h)$, for any $1 \le h \le \sqrt{m}$. The node requires constant time to route its inputs.

We shall need some definitions. An (r, s) *concentrator graph* [21] is a directed acyclic graph with r inputs and $s \le r$ outputs such that any $k \le s$ inputs can be simultaneously connected to some k outputs by vertex-disjoint paths. An (r, s, α) *partial concentrator graph* is a directed acyclic graph with r inputs and $s \le r$ outputs and a constant $0 < \alpha < 1$ such that any $k \le \alpha s$ inputs can be simultaneously connected to some k outputs by vertex-disjoint paths.

Pinsker [21] and Pippenger [22] showed that (r, s) concentrator networks can be built with $O(r)$ components using probabilistic constructions, but they do not bound the depth of the graph, which we wish to be constant. Pippenger [23], however, uses another probabilistic argument to construct (r, s, α) partial concentrator graphs for sufficiently large r where $s = 2r/3$ and $\alpha = 3/4$. The partial concentrator graphs are bipartite (no intermediate vertices between inputs and outputs), every input has degree at most 6, and every output has degree at most 9. By pasting several of these graphs together, outputs to inputs, any constant ratio of concentration can be obtained in constant depth. For a given set of inputs, the paths through the graph can be set up in polynomial time using network flow techniques or by performing a sequence of matchings on each level of the graph.

We use a partial concentrator graph to construct a good concentrator switch. We simply make switching decisions at the inputs to each level. These decision bits can be interleaved with the address bits that specify the path of a message through the fat-tree. In order to use the off-line routing results from Section III, we treat the actual capacity of a channel as α times the number of wires, which changes the results by only a constant factor.

We now turn our attention to the physical structure of a fat-tree node. A node with m incident wires contains $O(m)$ components. The next theorem gives the physical volume necessary to wire the components.

Lemma 3: A set of m components and external wires can be wired together according to an arbitrary interconnection pattern to fit in a box whose side lengths are $O(h\sqrt{m})$, $O(h\sqrt{m})$, and $O(\sqrt{m}/h)$, for any $1 \le h \le \sqrt{m}$.

Proof: We need to use the fact that in two dimensions, any permutation of m inputs and m outputs can be routed in $O(m^2)$ area, which can be seen by considering a "crossbar" layout. Thus, in two dimensions, the wiring of the components and external wires can be accomplished by laying all components and external wires along a line and routing the permutation dictated by the interconnection.

The construction in three dimensions is essentially that of Leighton and Rosenberg [16]. In three dimensions, the external wires and components lie on a face of a box. Any permutation of m inputs and m outputs can be routed in a box of $O(m^{3/2})$ volume where each side has length $O(\sqrt{m})$. This proves the result of the theorem for constant h.

To extend the result, we use a result of Thompson [29] on converting a layout of height h into a layout of height 2. Consider slicing the box into slices of height h, and consider one such slice. If we expand each of the other two dimensions by a factor of h, the h layers can be superimposed, slightly offset from one another. Since this can be done with each of the slices simultaneously, the theorem follows. ■

We are now in a position to ascertain the cost of a fat-tree implementation based on the capacities of its channels. If the capacities of the fat-tree channels are determined arbitrarily, the analysis could be messy. For the fat-trees that will be used in universality results of Section VI, however, the channel capacities can be characterized by the capacity at the root. This section defines the channel capacities of a *universal* fat-tree and evaluates the hardware costs of an implementation. Without loss of generality, and for simplicity, we assume in this section that the number of connections to each processor in the fat-tree is 1.

Let FT be a fat-tree on n processors, and let C be the set of channels in FT. Consider each node to have a level number that is its distance to the root, and give each channel $c \in C$ the same level number as the node beneath it. Thus, for example, the root and the channel between the root and the external interface are both at level 0. The processors and the channels leaving them are at level lg n. If the channel at level 0 has capacity w, then we say that FT has *root capacity* w.

Definition: Let FT be a fat-tree on n processors with root capacity w where $n^{2/3} \le w \le n$. Then if each channel $c \in C$ at level k satisfies

$$\mathrm{cap}(c) = \min\left\{\left\lceil \frac{n}{2^k} \right\rceil, \left\lceil \frac{w}{2^{2k/3}} \right\rceil\right\},$$

we call FT a *universal fat-tree*.

The capacities of the channels of a universal fat-tree grow exponentially as we go up the tree from the leaves. Initially, the capacities double from one level to the next, but at levels closer than 3 $\lg(n/w)$ to the root, the channel capacities grow at the rate of $\sqrt[3]{4}$.

We can now determine the hardware required by a universal fat-tree.

Theorem 4: Let FT be a universal fat-tree on n processors with root capacity w where $n^{2/3} \le w \le n$. Then there is an implementation of FT in a cube of volume $v = O((w \lg(n/w))^{3/2})$ with $O(n \lg(w^3/n^2))$ components.

Proof: We first establish the component count. For a node at level $k \le 3 \lg(n/w)$, the number of components in the node is $O(w/2^{2k/3})$, and the number of nodes at level k is 2^k. Thus, the number of components in all levels between 0 and 3 $\lg(n/w)$ is

$$\begin{aligned}\sum_{k=0}^{3\lg(n/w)} 2^k O(w/2^{2k/3}) &= w \sum_{k=0}^{3\lg(n/w)} O(2^{k/3}) \\ &= O(n)\end{aligned}$$

since the largest term of the geometric series occurs when $k = 3\lg(n/w)$. Nearer the leaves, each level has about the same number of components. The total number in the levels between 3 $\lg(n/w)$ and $\lg n$ is

$$\sum_{k=3\lg(n/w)}^{\lg n} 2^k O(n/2^k) = O(n \lg(w^3/n^2)).$$

Thus, the number of components nearer the leaves of the fat-tree dominates.

The volume bound is somewhat more intricate to establish, but is essentially the unrestricted three-dimensional layout construction given by Leighton and Rosenberg [16]. The interested reader is referred to their paper. Similar divide-and-conquer layout strategies for two dimensions can be found in [3], [12], [14], [17], [18], [32]. ■

Theorem 4 gives the volume of a fat-tree in terms of its root capacity. For the universality results of Section VI, we shall be interested in the reverse.

Definition: Let FT be a universal fat-tree that occupies volume v and has root capacity $\Theta(v^{2/3}/\lg(n/v^{2/3}))$. Then FT is a *universal fat-tree of volume* v.

Remark: A universal fat-tree on n processors of volume v must satisfy $v = \Omega(n \lg n)$ and $v = O(n^{3/2})$ to be well defined. By modifying the definition of a universal fat-tree, the lower bound can be relaxed to $\Omega(n)$, which results in minor changes to the bounds quoted in the universality theorem of Section VI.

V. Decomposition Trees

The physical implementation of a routing network constrains the ability of processors in a parallel supercomputer to communicate with one another. The universality theorem from Section VI makes essentially one assumption about competing networks: at most $O(a)$ bits can pass through a surface of area a in unit time. This assumption can be brought to bear on an arbitrary portion of a routing network implementation through the use of *decomposition trees*, a refinement of the graph-theoretic notion of *separators* [19]. Similar results can be found in the VLSI theory literature. The results presented here generalize and greatly simplify some of the constructions in the literature, notably those in [3], [4], and [13]. The generalizations are necessary for the proof of the universality theorem.

A routing network R interconnecting a set P of processors has a $[w_0, w_1, \cdots, w_r]$ *decomposition tree* if the amount of information that can enter or leave the set P of processors from the outside world is at most w_0 bits per unit time; P can be partitioned into two sets P_0 and P_1 such that the amount of information that can enter or leave each set is at most w_1 bits per unit time; each of P_0 and P_1 can be partitioned into two sets such that the bandwidth to and from each of the four sets is at most w_2; and so on, until every set at the rth level has either zero or one processors in it. When the bandwidth decreases by a constant amount from one level to the next, we shall adopt a shorthand notation. We shall say that R has a (w, α) decomposition tree for $1 < \alpha \leq 2$ if it has a $[w, w/\alpha, w/\alpha^2, \cdots, O(1)]$ decomposition tree. (For VLSI graph layouts, there is a similar notion called *bifurcators* [3], [13].)

Theorem 5: Let R be a routing network that occupies a cube of volume v. Then R has an $(O(v^{2/3}), \sqrt[3]{4})$ decomposition tree.

Proof: The cube has side length $\sqrt[3]{v}$ and surface area $6v^{2/3}$. Imagine a rectilinearly oriented plane that splits the cube into two equal boxes, each occupying volume $v/2$. This cutting plane naturally partitions the processors into two sets. Partition each of the two boxes by repeating this procedure with a plane perpendicular to the first. Continuing now in the third dimension yields eight cubes. Repeat this procedure until each box contains either zero or one processors.

The volume of each of the 2^i boxes generated by the ith cut is $v/2^i$, and the surface area is at most $4\sqrt[3]{4}\,(v/2^i)^{2/3}$. Let γ be the constant factor by which the bandwidth of information transfer differs from the surface area. Then the routing network R has a $(4\sqrt[3]{4}\,\gamma v^{2/3}, \sqrt[3]{4})$ decomposition tree. ■

A decomposition tree generated by the cutting plane method can be unbalanced in the sense that the number of processors lying on either side of a given cut may be unequal. Following the approach of Bhatt and Leighton [3], we define a *balanced decomposition tree* to be a decomposition tree in which the number of processors on either side of a given partition is equal, to within one. We shall show that a balanced decomposition tree can be produced from an unbalanced one.

First, however, we shall need two combinatorial lemmas. The first, which deals with the partitioning of strings of pearls, is typical of lemmas proved in the VLSI theory literature.

Lemma 6: Consider any two strings composed of even numbers of black and white pearls. By making at most two cuts, the pearls can be divided into two sets, each containing at most two strings, such that each set has exactly half the pearls of each color.

Proof:[2] Call the strings L and S for "long" and "short." We use a continuity argument to show that two sets A and $\overline{A}$ satisfying the conditions of the lemma can always be produced. Place the strings L and S end-to-end in a circle, as is illustrated in Fig. 4(a). Let A be the set of pearls comprising the shaded half of the circle in Fig. 4(b), and let $\overline{A}$ be the set of pearls in the other half. Suppose without loss of generality that the set A contains too many black pearls and set $\overline{A}$ contains too few. We shall show how to transform set A so that it occupies the initial position of set $\overline{A}$. The transformation consists of a sequence of moves such that for each

[2] Thanks to G. Miller of USC, who provided this argument, which is simpler than our original algebraic proof.

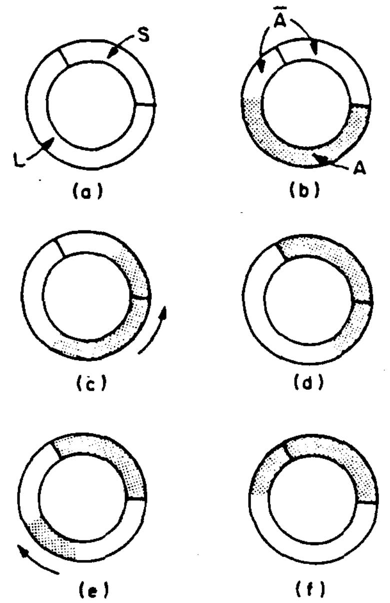


Fig. 4. The partitioning argument. (a) The two strings L and S laid end-to-end in a circle. (b) The initial position of set A. (c)–(f) The transformation of A into $\overline{A}$.

move, the number of blacks within set A changes by at most one. Since set A starts out with too many black pearls and ends with too few, by continuity there will be a position in the middle where A has exactly half the black pearls. Furthermore, because A has half the total number of pearls, it will also have half the white pearls.

The transformation begins by rotating set A counterclockwise, as shown in Fig. 4(c), until it reaches the position shown in Fig. 4(d). Then, set A is broken into two pieces and the tailing piece is rotated clockwise until it meets up with the leading piece on the other side, shown in Fig. 4(e) and (f). The position of set A is now the initial position of set $\overline{A}$. As can be verified, at all times during the transformation, sets A and $\overline{A}$ each contain at most two strings. ■

Lemma 7: Let T be a complete binary tree drawn in the natural way with leaves on a straight line, and consider any string s of k consecutive leaves. Then there exists a forest F of complete binary subtrees of T such that 1) the leaves of F are precisely the leaves in s, 2) there are at most two trees of any given height, and 3) the height of the largest tree is at most $\lg k$.

Proof: The forest is constructed from the maximal complete subtrees of T whose leaves lie only in s. ■

Theorem 8: Let R be a routing network on n processors that has a $[w_0, w_1, \cdots, w_r]$ decomposition tree T. Then R has a $[w'_0, w'_1, \cdots, w'_{\lceil \lg n \rceil}]$ balanced decomposition tree T' where

$$w'_i = 4 \sum_{k=i}^{r} w_k .$$

Proof: Draw the decomposition tree T in the natural way with the 2^r leaves on a line. Each leaf either contains a processor or else it is empty. If the leaf contains a processor, color it black; otherwise, color it white. Considering the line of processors as a string of black and white pearls, as in Lemma 6, we can cut the string in at most two places such that the pearls are divided into two sets, each containing at most two strings, such that each set has exactly half the pearls of each color. This partition represents the first level in the balanced decomposition tree T'.

Recursively partition each of the two sets using Lemma 6. At each step, the number of black pearls (processors) is split evenly in a set, and each set contains at most two strings (consecutive leaves from the decomposition tree T). Thus, at level $\lceil \lg n \rceil$, each set (leaf of T') contains at most one processor.

It remains to prove the bound on the rates w'_i of information transfer in and out of each subtree of the balanced decomposition tree T'. Each subtree of T' corresponds to at most two strings of leaves from the original decomposition tree T. From Lemma 7, these two strings correspond to a forest of complete binary trees with at most four trees of a given height. All external communication of a complete binary subtree of a decomposition tree occurs through the surface corresponding to its root. Thus, the external communication per unit time of a subtree of T' is bounded by the sum of bandwidths from the roots of the corresponding complete binary subtrees of T. ■

Corollary 9: Let R be a routing network that has a (w, a) decomposition tree for $1 < \alpha \leq 2$. Then S has a $(4(\alpha/\alpha - 1)w, \alpha)$ balanced decomposition tree.

Proof: The summation in Theorem 8 becomes a geometric series. ■

VI. Universality of Fat-Trees

We now show that a fat-tree is universal for the amount of interconnection hardware it requires in the sense that any other routing network of the same volume can be efficiently simulated. From a theoretical point of view, we define "efficiently" as meaning at most polylogarithmic slowdown. Polylogarithmic time in parallel computation corresponds to polynomial time for sequential computation.

Some may argue that polylogarithmic slowdown may not be efficient if the exponent of the logarithm is large. The ability of one parallel computer to simulate another, however, merely gives confidence in the general-purpose nature of the computer. The loss of efficiency in the simulation is not felt if the parallel computer is programmed directly.

Many of the networks currently being built are not universal (for example, two-dimensional arrays, simple trees, or multigrids). These networks exhibit polynomial slowdown when simulating other networks. Thus, they have no theoretical advantage over a sequential computer which can easily simulate a network with polynomial slowdown. Interestingly, hypercube-based networks are universal for volume $\Theta(n^{3/2})$, but as we have observed, they do not scale down to smaller volumes.

Theorem 10: Let FT be a universal fat-tree on a set of n processors that occupies a cube of volume v, and let R be an arbitrary routing network on a set of n processors that also occupies a cube of volume v. Then there is an identification *of the processors in FT with the processors of R with the following property. Any message set M that can be delivered in time t by R can be delivered by FT (off-line) in time* $O(t \lg^3 n)$.

Proof: By Theorem 5, the routing network R has an $(O(v^{2/3}), \sqrt[3]{4})$ decomposition tree, and hence by Corollary 9, it also has an $(O(v^{2/3}), \sqrt[3]{4})$ balanced decomposition tree. Identify the processors at the leaves of the balanced decomposition tree of R in the natural way, with the processors at the leaves of the fat-tree FT.

By assumption, routing network R can deliver all the messages in a message set M in time t. In unit time, at most $O(v^{2/3}/2^{2k/3})$ messages can enter or leave a subtree rooted at level k in R's balanced decomposition tree. Thus, in t time, the total number of messages that can enter or leave a subtree rooted at level k is $O(tv^{2/3}/2^{2k/3})$.

There is a second bound on the transfer of information in and out of a subtree of the balanced decomposition tree of R. The number of messages that can enter or leave a single processor in time t is $O(t)$ since the number of connections to a processor is constant. Since there are at most $n/2^k$ processors in a subtree rooted at level k, the total number of messages that can enter or leave this subtree in t time is $O(tn/2^k)$.

We now compute an upper bound on the load factor $\lambda(M)$ that M puts on the fat-tree FT. Let c be a channel at level k in FT. We have just seen that the number load(M, c) of messages of M that must go through c is $O(tv^{2/3}/2^{2k/3})$ and $O(tn/2^k)$. Since FT is a universal fat-tree with root capacity $\Theta(v^{2/3}/\lg(n/v^{2/3}))$, the capacity of c is

$$\operatorname{cap}(c) = \min\left\{\left\lceil \frac{n}{2^k} \right\rceil, \Theta\left(\frac{v^{2/3}}{2^{2k/3}\lg(n/v^{2/3})}\right)\right\}.$$

Thus, the load factor on c due to M is

$$\lambda(M, c) = O(t \lg(n/v^{2/3})),$$

and the load factor on the whole fat-tree is

$$\lambda(M) = O(t \lg(n/v^{2/3})).$$

The off-line routing result from Theorem 1 says that $O(t \lg(n/v^{2/3}) \lg n)$ delivery cycles are sufficient to route all the messages in M. Since the fat-tree can execute an off-line delivery cycle in $O(\lg n)$ time, the result follows. ■

The $O(\lg^3 n)$ factor lost in simulation is attributable to the channel capacities, the routing algorithm, and the switching. Of these three, only the last, the $O(\lg n)$ switching time for a delivery cycle, seems to be a necessary cost.

The first $O(\lg n)$ factor (actually $O(\lg(n/v^{2/3}))$) is because a fat-tree of volume v has a root capacity of $O(v^{2/3}/\lg(n/v^{2/3}))$. This logarithmic factor vanishes for the simulation of networks that have only slightly less $(O(v/\lg^{3/2}(n/v^{2/3})))$ volume. We have chosen to put all the simulation expense in time so that the comparison will be equal hardware versus equal hardware.

The second $O(\lg n)$ factor is lost by the off-line routing algorithm. In fact, we have recently discovered [8] that off-line routing in $O(\lambda(M) + \lg n \lg \lg n)$ delivery cycles is always possible. Moreover, if we assume that each processor has $\Theta(\lg n)$ connections, as is required by a Boolean hypercube, and each channel has capacity $\Omega(\lg n)$, Corollary 2 from Section III allows us to route in $O(\lambda(M))$ delivery cycles.

An important application of the universality of fat-trees is to the simulation of *fixed-connection networks*, that is, networks that have direct connections between processors. Here we relax the technical assumption in the definition of a universal fat-tree to allow the processors to have a given number d of connections to the routing network, instead of 1. Such a universal fat-tree of volume $O(v \lg^{3/2}(n/v^{2/3}))$ on n processors can simulate an arbitrary degree d fixed-connection network of volume v on n processors with only $O(\lg n)$ time degradation. The idea is that the channel capacities of the universal fat-tree are sufficiently large that the connections implied by the network can be represented as a one-cycle message set, which requires $O(\lg n)$ time to be delivered.

High-volume universal fat-trees can be compared to classical permutation networks, which all require $\Omega(n^{3/2})$ volume. A universal fat-tree on n processors with $O(n^{3/2})$ volume can route an arbitrary permutation off-line in time $O(\lg n)$. Up to constant factors, this is the best possible bound (assuming bounded-degree processors), but it is also achievable, for instance, by Benes networks [2], [34] or by on-line sorting networks [1], [15].

A natural extension to the off-line routing results presented here, and indeed, the one that motivates the entire paper, is the problem of on-line routing in fat-trees. Not surprisingly, there are universal fat-trees for on-line routing. In results to be reported elsewhere [8] we have discovered a randomized routing algorithm that delivers all messages in $O(\lambda(M) + \lg n \lg \lg n)$ delivery cycles with high probability [8], but the nodes of the fat-tree have somewhat different structure from the design given here. Using this result and essentially the construction given in this paper, one can obtain an on-line analog to Theorem 10, except with an $O(\lg^3 n \lg \lg n)$ time degradation. We anticipate further research will improve this bound.

VII. Concluding Remarks

Universality has been studied more generally in the parallel computation literature. Valiant [33] and Valiant and Brebner [31] have discovered universal routing schemes for large-volume networks. Galil and Paul [7] have proposed a general-purpose parallel processor based on the cube-connected-cycles network [25] that can simulate any other parallel processor with only a logarithmic loss in efficiency. Valiant [30] has shown that there are classes of universal Boolean circuits. A universal circuit of a given size can be programmed to simulate any circuit whose size is only slightly smaller. Fiat and Shamir [6] have proposed a universal architecture for systolic array interconnections.

Universal fat-trees are parameterized not only in the number of processors, but also in volume, which is indirectly a measure of communication potential. By considering arbitrary networks in terms of these two parameters, we have seen that the one fat-tree architecture is near-optimal throughout the entire range of the parameters. For communication-limited engineering situations, one need not necessarily retreat to special-purpose devices in order to compute efficiently in parallel.

Fat-trees have the advantage that they are a robust engineering structure. In principle, one need not worry about the exact capacities of channels as long as the capacities exhibit reasonable growth as we go up the tree. As a practical matter, one should build the biggest fat-tree that one can afford, and the architecture automatically ensures that communication bandwidth is effectively utilized. Another feature of fat-trees is that algorithms are the same no matter how big the fat-tree is. Code is portable in that it can be moved between an inexpensive computer and a more expensive one. Finally, the root channel offers a natural high-bandwidth external connection.

Although universal fat-trees have many desirable properties, there are many issues in the design of a routing network that we have not faced directly. For example, despite our concern for wirability, we have not presented a practical packaging scheme. Possibly, the packaging techniques for trees from [4] and [18] can be exploited. The constraints to be faced in packaging, however, will only be more stringent than the surface area constraint given in Assumption L3. We have attempted to deal with "pin boundedness" in a simple mathematical model, and our results should generalize to more complicated packaging models.

Another issue that we have not addressed is how messages should be sent in the network. The choice of the bit-serial approach in Section II has the advantage that the hardware is cheaper, but we may be paying in the performance of the routing algorithm. We also assumed the architecture was synchronized by delivery cycle. Presumably, fat-tree architectures can be built with different design decisions.

Whether the notion of universal parallel supercomputers is consistent with engineering reality, however, remains an open question. Independent of routing network issues, there are many other problems that must be solved if abstract n-processor parallel supercomputers are to become a reality. For example, problems of maintenance, fault tolerance, clock distribution, and reliable power supply must be solved. The hardware mechanisms needed for synchronization and instruction distribution, which are simple for single-processor machines, may be sufficiently complicated to overwhelm the advantages of having many processors.

But the largest problem that must be solved in parallel supercomputing seems to us to be the problem of programming the system with the concerns of both programming abstraction and algorithmic integrity (computational resources are not free). A supercomputer should not be a mere supercalculator (good at one restricted algorithm). It should have the power to efficiently execute many different parallel algorithms and to easily combine the results of separate parallel computations. A universal machine has the power, not just of any other machine, but of all other machines.

ACKNOWLEDGMENT

T. Knight of the M.I.T. Artificial Intelligence Laboratory inspired the author with many discussions about connection machine architectures, which led him towards the ideas in this paper. He is to blame for the name "fat-tree." Thanks to R. Greenberg of the M.I.T. Laboratory for Computer Science for finding many bugs in various versions of the paper. Thanks also to A. Bawden, T. Bui, B. Chor, T. Cormen, O. Goldreich, A. Ishii, T. Leighton, B. Maggs, M. Maley, G. Miller, C. Phillips, R. Rivest, and P. Shor for their helpful comments and technical assistance.

REFERENCES

[1] M. Ajtai, J. Komlos, and E. Szemeredi, "Sorting in c log n parallel steps," *Combinatorica*, vol. 3, no. 1, pp. 1–19, 1983.

[2] V. E. Benes, *Mathematical Theory of Connecting Networks and Telephone Traffic*. New York: Academic, 1965.

[3] S. N. Bhatt and F. T. Leighton, "A framework for solving VLSI graph layout problems," *J. Comput. Syst. Sci.*, vol. 28, no. 2, pp. 300–343, Apr. 1984.

[4] S. N. Bhatt and C. E. Leiserson, "How to assemble tree machines," in *Advances in Computing Research, Vol. 2*. Greenwich, CT: Jai Press, 1984, pp. 95–114.

[5] S. Fahlman, *NETL: A System for Representing and Using Real-World Knowledge*. Cambridge, MA: M.I.T. Press, 1979.

[6] A. Fiat and A. Shamir, "Polymorphic arrays: A novel VLSI layout for systolic computers," in *Proc. IEEE 25th Annu. Symp. Foundations Comput. Sci.*, Oct. 1984, pp. 37–45.

[7] Z. Galil and W. J. Paul, "An efficient general-purpose parallel computer," *J. ACM*, vol. 30, no. 2, pp. 360–387, Apr. 1983.

[8] R. I. Greenberg and C. E. Leiserson, "Randomized routing on fat-trees," in *Proc. IEEE 26th Annu. Symp. Foundations Comput. Sci.*, Nov. 1985, to appear.

[9] W. D. Hillis, "The connection machine," Artif. Intell. Lab., Mass. Inst. Technol., Tech. Memo. 646, Sept. 1981.

[10] R. M. Karp, F. T. Leighton, R. L. Rivest, C. D. Thompson, U. Vazirani, and V. Vazirani, "Global wire routing in two-dimensional arrays," in *Proc. IEEE 24th Annu. Symp. Foundations Comput. Sci.*, Nov. 1983, pp. 453–459.

[11] T. F. Knight, "The cross-omega router," Mass. Inst. Technol., Cambridge, unpublished manuscript, 1984.

[12] F. T. Leighton, "New lower bound techniques for VLSI," *Math. Syst. Theory*, vol. 17, no. 1, pp. 47–70, Apr. 1984.

[13] ——, "A layout strategy for VLSI which is provably good," in *Proc. 14th Annu. ACM Symp. Theory Comput.*, May 1982, pp. 85–98.

[14] ——, *Complexity Issues in VLSI*. Cambridge, MA: M.I.T. Press, 1983.

[15] ——, "Tight bounds on the complexity of parallel sorting," *IEEE Trans. Comput.*, vol. C-34, pp. 344–354, Apr. 1985.

[16] F. T. Leighton and A. L. Rosenberg, "Three dimensional circuit layouts," Lab. Comput. Sci., Mass. Inst. Technol., Tech. Rep. MIT-LCS-TM-262, June 1984.

[17] C. E. Leiserson, "Area-efficient graph layouts (for VLSI)," in *Proc. IEEE 21st Annu. Symp. Foundations Comput. Sci.*, Oct. 1980, pp. 270–281.

[18] ——, *Area-Efficient VLSI Computation*. Cambridge, MA: M.I.T. Press, 1983.

[19] R. J. Lipton and R. E. Tarjan, "A planar separator theorem," *SIAM J. Appl. Math.*, vol. 36, no. 2, pp. 177–189, Apr. 1979.

[20] C. A. Mead and L. A. Conway, *Introduction to VLSI Systems*. Reading, MA: Addison-Wesley, 1980.

[21] M. S. Pinsker, "On the complexity of a concentrator," in *Proc. 7th Int. Teletraffic Conf.*, Stockholm, Sweden, June 1983, pp. 318/1–318/4.

[22] N. J. Pippenger, "The complexity theory of switching networks," Ph.D. dissertation, Dep. Elec. Eng. Comput. Sci., Mass. Inst. Technol., Cambridge, Aug. 1973.

[23] ——, "Superconcentrators," *SIAM J. Comput.*, vol. 6, no. 2, pp. 298–304, June 1977.

[24] ——, "Parallel communication with limited buffers," in *Proc. IEEE 25th Annu. Symp. Foundations Comput. Sci.*, Oct. 1984, pp. 127–136.

[25] F. P. Preparata and J. E. Vuillemin, "The cube-connected cycles: A versatile network for parallel computation," *Commun. ACM*, vol. 24, no. 5, pp. 300–309, May 1981.

[26] A. L. Rosenberg, "Three-dimensional integrated circuitry," in *Proc. CMU Conf. VLSI Syst. Comput.*, H. T. Kung, R. Sproull, and G. Steele, Eds., Oct. 1981, pp. 69–79.

[27] J. T. Schwartz, "Ultracomputers," *ACM Trans. Programming Lang. Syst.*, vol. 2, no. 4, pp. 484–521, 1980.

[28] H. S. Stone, "Parallel processing with the perfect shuffle," *IEEE Trans. Comput.*, vol. C-20, pp. 153–161, Feb. 1971.

[29] C. D. Thompson, "A complexity theory for VLSI," Ph.D. dissertation, Carnegie-Mellon Univ., Pittsburgh, PA, 1980.

[30] L. G. Valiant, "Universal circuits," in *Proc. 8th Annu. ACM Symp. Theory Comput.*, May 1976, pp. 196–203.

[31] L. G. Valiant and G. J. Brebner, "Universal schemes for parallel communication," in *Proc. 13th Annu. ACM Symp. Theory Comput.*, May 1981, pp. 263–277.

[32] L. G. Valiant, "Universality considerations in VLSI circuits," *IEEE Trans. Comput.*, vol. C-30, pp. 135–140, Feb. 1981.

[33] ——, "A scheme for fast parallel communication," *SIAM J. Comput.*, vol. 11, no. 2, pp. 350–361, May 1982.

[34] A. Waksman, "A permutation network," *J. ACM*, vol. 15, no. 1, pp. 159–163, Jan. 1968.

Charles E. Leiserson (M'83) received the B.S. degree in computer science and mathematics from Yale University, New Haven, CT, in 1975 and the Ph.D. degree in computer science from Carnegie-Mellon University, Pittsburgh, PA, in 1981.

He is currently an Associate Professor of Computer Science and Engineering at the Massachusetts Institute of Technology (M.I.T.), Cambridge. In 1981 he joined the faculty of the Theory of Computation Group in the M.I.T. Laboratory for Computer Science. His expertise includes parallel computation, VLSI architectures, graph theory, digital circuit timing, analysis of algorithms, computer-aided design, placement and routing, wafer-scale integration, layout compaction, and most recently, parallel supercomputing. His principal interest is in the theoretical foundation of parallel computation, especially as it relates to engineering reality.

Prof. Leiserson has authored over 20 papers on the theory of VLSI and parallel algorithms. As a graduate student at Carnegie-Mellon he wrote the first paper on systolic architectures with H. T. Kung, for which they received a U.S. patent. His Ph.D. dissertation, "Area-Efficient VLSI Computation," which deals with the design of systolic systems and with the problem of determining the VLSI area of a graph, won the first ACM Doctoral Dissertation Award. He was awarded a Presidential Young Investigator Award in 1985. He is a member of the ACM, and he serves on the ACM General Technical Achievement Award Committee, which selects the Turing Award winner.

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

VLSI PUBLICATIONS

VLSI Memo No. 85-278
October 1985

SIGNAL DELAY IN LEAKY RC MESH MODELS FOR BIPOLAR INTERCONNECT

Peter O'Brien and John L. Wyatt, Jr.

Abstract

This paper extends the results of Penfield and Rubinstein on signal delay in RC tree networks. Both estimates and bounds are derived for the step response of "leaky" RC trees and meshes, a class of networks that is appropriate for modelling interconnect in digital bipolar circuits. This paper is intended to serve as a tutorial as well as a research report. Therefore existing work is explained in some detail along with the derivation of new results.

Microsvstems
Research Center
Room 39-321

Massachusetts
Institute
of Technology

Cambridge
Massachusetts
02139

Teleohone
(617) 253-8138

SIGNAL DELAY IN ECL INTERCONNECT

Peter R. O'Brien, J. L. Wyatt, Jr.

Department of Electrical Engineering and Computer Science,
Massachusetts Institute of Technology, Cambridge, MA 02139

Abstract

Rubinstein et. al. [1] have derived rigorous, closed-form bounds on the step response of linear, nonuniform RC trees. Wyatt [2] has extended these bounds to include RC meshes. These results have proven to be useful for fast estimation and bounding of interconnect delay in digital MOS integrated circuits [3-5]. We further extend these results to include leaky RC trees and meshes, a class of networks that is appropriate for modelling interconnect in digital bipolar circuits. Finally, we discuss two practical problems involved in using these results: 1.) Extra computational requirements. 2.) Accurate modelling of digital bipolar gates.

I. Introduction

A leaky RC tree or mesh meets all of the requirements of an RC tree or mesh (as described in [1] and [2]), except for the restriction that there be no grounded resistors. The practical motivation behind allowing resistive paths to ground in RC mesh circuits is an attempt to more accurately model interconnect on bipolar chips. The base of a bipolar transistor loading the interconnect offers a (nonlinear) resistive path to ground. A corresponding gate electrode of an MOS transistor would not offer such a resistive path to ground, and previous works (e.g., [1],[2]) relied on the abscence of such paths.

This paper is concerned with bounds on the step response of a linear, lumped, leaky RC tree or mesh driven by an ideal voltage source, and starting from some arbitrary (possibly non-zero) initial equilibrium condition.

II. Network Differential Equations for RC Meshes

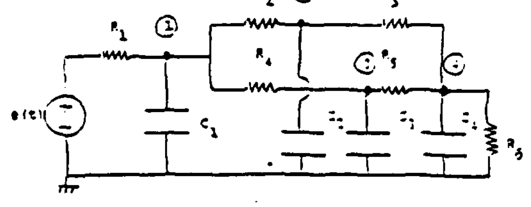


Fig. 1: Leaky RC mesh.

Isolate the resistor subnetwork $\mathcal{R}$ containing all the resistors and assign reference directions to the capacitor currents as shown in Fig. 2. Let the node connected to the independent source serve as the datum node of $\mathcal{R}$. With the datum grounded, the node voltages are expressed in terms of the capacitor currents by the positive-definite, symmetric matrix $\underset{\sim}{R}$ as shown below.

$$\begin{bmatrix} v_1 \\ \vdots \\ v_N \end{bmatrix} = \begin{bmatrix} R_{11} & \cdots & R_{1N} \\ \vdots & & \vdots \\ R_{N1} & \cdots & R_{NN} \end{bmatrix} \begin{bmatrix} i_1 \\ \vdots \\ i_N \end{bmatrix} \qquad (1)$$

When e(t) is not zero, then by superposition we have:

$$\underset{\sim}{v}(t) = \underset{\sim}{R}\underset{\sim}{i}(t) + \underset{\sim}{x}e(t) \qquad (2)$$

where $\underset{\sim}{x} = (x_1, \ldots, x_N)^T$ is a column vector of dimensionless numbers, each x_i being numerically equal to the potential (in volts) produced at node i due solely to a 1-volt source at the datum, while open-circuiting all of the external current sources.

Consider the response to a step change in e(t). Substituting $i_k = -C_k\dot{v}_k$ into (2), and identifying $v_{i_{eq}} = x_i e(\infty) = x_i e(t)$ as the final equilibrium voltage at node i, yields the network differential equations

$$v_{i_{eq}} - v_i(t) = \sum_{j=1}^{N} R_{ij}C_j\dot{v}_j(t);\ i=1,\ldots,N;t>0 \qquad (3)$$

which are identical in form to eq. (2) of [2]: the only differences are that in [2] $v_{i_{eq}} = 1$ for all nodes i, and the resistance matrix $\underset{\sim}{R}=(R_{ij})$ does not reflect the presence of any resistive paths to ground.

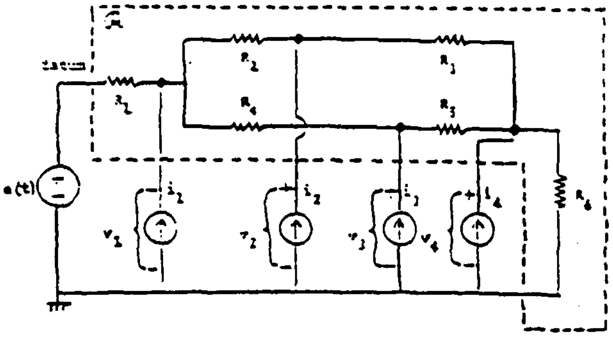


Fig. 2: The resistor subnetwork R extracted from the circuit in Fig. 1 (driven with current sources)

III. Optimal Control Method for Determining Step Response Bounds

The derivation outlined below is similar to that in [6], but the results now apply to leaky meshes starting from an arbitrary (possibly nonzero) equilibrium condition. We shall use the notation $\underset{\sim}{v}_o$ to denote the <u>initial equilibrium voltage distribution</u>, and $\underset{\sim}{v}_{eq}$ to denote the *final* <u>equilibrium voltage distribution</u>.

<u>Fact 1</u>

For any three nodes, i,j,k of a leaky RC mesh,

$$R_{ii}R_{kj} \geq R_{ki}R_{ij} \qquad (4)$$

where the resistances in (4) are elements of the resistance matrix $\underset{\sim}{R}$ in (1).

The proof given in [2] for non-leaky RC meshes generalizes to leaky RC meshes in a straightforward way (for details, see also [7]).

<u>Fact 2</u>

The step response of a leaky RC mesh is completely monotone, i.e., for any node j we have:

$$\dot{v}_j(t) \geq 0, \ \forall t \geq 0; \text{ if } v_{j_{eq}} > v_{j_o} \qquad (5)$$
$$\dot{v}_j(t) \leq 0, \ \forall t \geq 0; \text{ if } v_{j_{eq}} < v_{j_o}$$

The proof is in [7].

<u>Fact 3</u>

For any two nodes i, k of a leaky RC mesh: On an "up" ("down") transition, i.e., $\underset{\sim}{v}_{eq} > \underset{\sim}{v}_o$ ($\underset{\sim}{v}_{eq} < \underset{\sim}{v}_o$), we have

$$R_{ii}\left[v_{k_{eq}} - v_k(t)\right] \geq (\leq) \ R_{ki}\left[v_{i_{eq}} - v_i(t)\right] \qquad (6)$$

$$R_{ki}\left[v_{k_{eq}} - v_k(t)\right] \leq (\geq) \ R_{kk}\left[v_{i_{eq}} - v_i(t)\right] \qquad (7)$$

Given Facts 1 and 2, Fact 3 follows from the network differential equations (3). See [7] for a detailed derivation.

We now construct a reduced-order model, as in [6]. Choosing a distinguished node i as the output node of interest, we describe the system in terms of only two state variables, the "normalized distance to go"

$$g_i(t) \triangleq \frac{v_{i_{eq}} - v_i(t)}{v_{i_{eq}} - v_{i_o}} \qquad (8)$$

and its integral

$$f_i(t) \triangleq \int_t^\infty g_i(t')dt' = \frac{\sum_k R_{ik}C_k(v_{k_{eq}} - v_k(t))}{v_{i_{eq}} - v_{i_o}} \qquad (9)$$

where the last equality follows upon substituting (3) into the integral and evaluating. Using Fact 3 in (9) yields the state constraints:

$$T_{R_i}g_i(t) \leq f_i(t) \leq T_P g_i(t) \qquad (10)$$

where $T_{R_i} \triangleq \frac{1}{R_{ii}} \sum_k R_{ik}^2 C_k$ and $T_P \triangleq \sum_k R_{kk}C_k$.

The initial condition on $g_i(t)$ is obviously

$$g_i(0) = 1 \qquad (11)$$

The initial condition on $f_i(t)$ is:

$$f_i(0) = \frac{\sum_k R_{ik}C_k(v_{k_{eq}} - v_{k_o})}{v_{i_{eq}} - v_{i_o}} = \frac{1}{x_i}\sum_k R_{ik}C_k x_k \triangleq T_{D_i} \qquad (12)$$

Note that in the non-leaky case, $x_k = 1 \ \forall k$, and so T_{D_i} reduces to $\sum_k R_{ik}C_k$ (which agrees with its definition in earlier works [1], [2]).

Hence, the optimal control problem of maximizing (minimizing) the time for $g_i(t)$ to reach a "target" value of $0<g_i^*\leq 1$ subject to the state constraints (10), initial conditions (11) and (12), and dynamics:

$$\begin{bmatrix} \dot{f}_i(t) \\ \dot{g}_i(t) \end{bmatrix} = \begin{bmatrix} 0 & -1 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} f_i(t) \\ g_i(t) \end{bmatrix} + \begin{bmatrix} 0 \\ 1 \end{bmatrix} u(t);$$

$$u(t) \leq 0 \qquad (13)$$

is algebraically equivalent to the problem in [6] and results in upper (lower) bounds on $g_i(t)$ [corresponding to lower (upper) bounds on $v_i(t)$]. The exact expressions are omitted for the sake of brevity, and they agree with prior results in [1] and [2] for non-leaky meshes. The complete bound expressions, along with a detailed derivation can be found in [7].

The fundamental reason why the bounding approach presented in this section carries over so successfully from non-leaky to leaky meshes is that in each case, the dynamics of the circuit is governed by a (negative) "M-matrix". [8]

IV. Extra Computational Requirements

The formulas for the time constants T_{R_i}, T_{D_i}, and T_P indicate that the information required in order to compute bounds for leaky RC meshes is as follows:

1.) The i^{th} row (or column) of the resistance matrix $\underset{\sim}{R}$.

2.) The diagonal elements of the resistance matrix $\underset{\sim}{R}$.

3.) The vector $\underset{\sim}{x}$ discussed in section II.

If, as will be the case in our model, the RC network has only one capacitor node (node #1) connected to the driving voltage source through a single resistor (R_S), then it can be shown [7] that $x_k = R_{k1}/R_S \forall k$. So, in fact, item 3) above is really equivalent to the first row(or column) of the resistance matrix $\underset{\sim}{R}$.

Hence, the computational problem consists of finding certain elements of the resistance matrix R. Unfortunately, the R matrix is "global" in character and is difficult to compute if there are resistor loops or resistive paths to ground (as opposed to the case of non-leaky RC trees in [1], where the desired elements of $\underset{\sim}{R}$ could be determined very easily).

However, the conductance matrix $\underset{\sim}{G} = \underset{\sim}{R}^{-1}$ is "local" in character, sparse, and can be determined virtually by inspection even when there are resistor loops or grounded resistors. One approach, then, is to determine G directly and invert it to find $\underset{\sim}{R}$. An efficient numerical algorithm to do this is the iterative "conjugate gradient" method [9; chapter 10] which exploits the fact that G is sparse, symmetric, and positive-definite. This algorithm would have to be applied N times to solve the linear systems:

$$\underset{\sim}{G}(\underset{\sim}{R})_i = \underset{\sim}{e}_i \; ; \; i = 1, \ldots, N \qquad (14)$$

Alternate approaches to computing resistance matrices have been explored by Brock [10]. One such algorithm is recursive and involves removing resistors one at a time until the original resistor subnetwork is reduced to a non-leaky tree. The efficiency of this algorithm depends on the number of resistors that must be removed. Another such algorithm is direct and involves a fixed cost of enumerating all possible spanning trees of the original resistor subnetwork. Then, only the elements of $\underset{\sim}{R}$ that are desired (which is an advantage in our application) are computed at a smaller additional cost per element.

V. Modelling of Gates

Accurate modelling of ECL gates is a crucial prerequisite to applying the theory in this paper to the timing analysis of real bipolar digital circuits. Both the sourcing and loading effects of the gates must be considered. As of this writing, the sourcing effects have not yet been modelled, but the loading effects have. We present here the results of our load modelling, and a strategy for source modelling. The gate model we are considering is shown below in Fig 3.

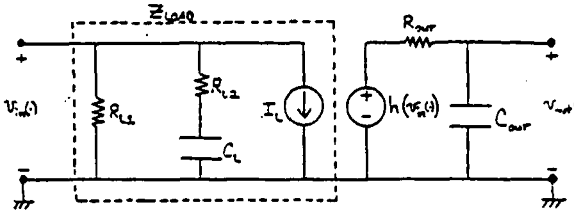


Fig. 3: Proposed Model of ECL Gate

We have determined load impedance parameters of R_{L1} = 120kΩ, R_{L2} = 1.8kΩ, C_L = .065pF, and I_L = 13μA which match our particular SPICE model of an ECL gate very closely. We tested this load model by driving with the "real" SPICE gate over a wide range of interconnect runs (lumped RC lines from 0 to 500 mils; r = 40Ω/100 mils; c = .683pF/100 mils) and fanout (1 to 10 loads in parallel). Deviation from the true delay (measured from the source gate input to the load gate input) was less than 5%, the worst case being high-fanout with short interconnect. In most cases, especially with lower fanout, errors were substantially less than 5%. While not explicitly mentioned in the previous sections, the current source I_L, since its value never changes, does not complicate the bounding theory. It merely provides a necessary dc offset, since our ECL gates swing between -1.55 volts (low) and -1.05 volts (high).

The source modelling is somewhat more complicated. First, it is clear that two different source models are needed since there is a definite asymmetry between rising and falling transitions. Our SPICE model of an ECL gate can source more current than it can sink, so delays are greater and waveform slew rates are lower for a falling transition. Second, it is clear that a (nonlinear) mapping from $v_{in}(.)$ to $h(v_{in}(.))$ is needed. We expect the mapping h to be close to the following simple form: If $v_{in}(.)$ is characterized by a starting time of t=0 and a slew rate $T_{S_{in}}$, then $h(v_{in}(.))$ is characterized by a starting time of t = T_D and a slew rate of $T_{S_{out}}$, i.e., h will have both a pure delay effect and a simple waveshape effect. Finally, we note that the response to a non-step driving waveform, such as $h(v_{in}(.))$, can still be bounded in a manner consistent with the theory presented in this paper. Since the leaky RC mesh being driven is linear, bounding the step response is equivalent to bounding the response to an arbitrary non-step input [1],[7].

Acknowledgement

We gracefully acknowledge many helpful conversations with Mr. Thomas Savarino of the Digital Equipment Corporation, Marlboro, Massachusetts, who first posed many of the questions addressed in this paper.

This research was supported by the National Science Foundation under Grant No. ECS83-10941 and the Defense Advanced Research Projects Agency under Contract No. N00014-80-C-0622.

References

[1] Rubinstein, J., P. Penfield, Jr., and M.A. Horowitz, "Signal Delay in RC Tree Networks," IEEE Trans. Computer-Aided Design, vol. CAD-2, no. 3, pp.202-211. July 1983.

[2] Wyatt, J.L., Jr., "Signal Delay in RC Mesh Networks," IEEE Trans. Circuits and Systems, vol. CAS-32, no. 5, pp.507-510, May 1985.

[3] Putatunda, R., "Autodelay: A Second-Generation Automatic Delay Calculation Program for LSI/VLSI Chips," Proc. IEEE Int. Conf. on Computer-Aided Design, Santa Clara, CA, November 1984, pp.188-190.

[4] Tamura, E., K. Ogawa, and T. Nakano, "Path Delay Analysis for Hierarchical Building Block Layout System," ACM IEEE 20th Design Autom. Conf. Proc., June 1983, pp.403-410.

[5] Jouppi, N., "Timing Analysis for NMOS VLSI," ACM IEEE 20th Design Autom. Conf. Proc., June 1983, pp.411-418.

[6] Yu, Q., J.L. Wyatt, Jr., C. Zukowski, H.N. Tan, and P. O'Brien, "Improved Bounds on Signal Delay in Linear RC Models for MOS Interconnect," Proc. 1985 IEEE Int. Symp. on Circuits and Systems, Kyoto, Japan, June 1985, pp.903-906.

[7] O'Brien, P. and J.L. Wyatt, Jr., "Signal Delay in Leaky RC Mesh Models for Bipolar Interconnect," VLSI Memo No. 85-278, November 1985. All VLSI Memos are available from the Microsystems Program Office, Room 39-321, Dept. of Electrical Engineering and Computer Science, MIT, Cambridge, MA 02139.

[8] J.L. Wyatt, Jr., C. Zukowski, and P.Penfield, Jr., "Step Response Bounds for Systems Described by M-matrices, with Application to Timing Analysis of Digital MOS Circuits," Proc. 24th IEEE Conf. on Decision and Control, Ft. Lauderdale, FL, December 1985, pp1552-1557.

[9] Golub, Gene H., and Van Loan, Charles F., Matrix Computations, The Johns Hopkins University Press, Baltimore, 1983.

[10] Brock, M.L. "Algebraic and Graph-Theoretic Aspects of Networks," S.M. Thesis, Massachusetts Institute of Technology, December 1984.

END

DTIC

5-86